FOREST: Focused Object Retrieval by Exploiting Significant Tag Paths

Marilena Oita, Pierre Senellart

FOREST: Focused Object Retrieval by Exploiting Significant Tag Paths

Marilena Oita
INRIA Saclay – Île-de-France
& Télécom ParisTech; CNRS LTCI
Paris, France
marilena.oita@telecom-paristech.fr

Pierre Senellart
Institut Mines–Télécom
Télécom ParisTech; CNRS LTCI
Paris, France
pierre.senellart@telecom-paristech.fr

ABSTRACT

Data-intensive Web sites, e.g., blogs or news sites, present pages containing Web articles (a blog post, a news item, etc.). These Web articles, typically automatically generated by a content management system, use a fixed template and variable content. Unsupervised extraction of their content (excluding the boilerplate of Web pages, i.e., their common template) is of interest in many applications, such as indexing or archiving. We present a novel approach for the extraction of Web articles from dynamic Web pages. Our algorithm, FOREST, targets the zone of the Web page relevant to some (automatically acquired) *keywords* for a Web page to obtain structural patterns identifying the content of interest. We consider two potential source of keywords: Web feeds that may link to the Web page, and terms found through a frequency analysis on the Web page itself. These structural patterns are aggregated among different Web pages that use the same layout, and ranked using a new measure of relevance with respect to the set of keywords. We extensively evaluate FOREST and report improved results over the state of the art in Web article content extraction.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Algorithms, Experimentation

Keywords

content extraction, keywords, structural similarity, tag path

1. INTRODUCTION

Textual Web content on modern Web sites is, in the overwhelming majority of cases, produced by dedicated content management systems (CMSs). Such software generates Web pages containing textual articles (news items, forum messages, wiki articles, blog posts, tweets, etc.) by filling a template with information fetched from databases. In this process, the original textual or structured content are turned into full-fledged HTML documents, where the Web article is hidden among the markup encoding the site layout. Some parts of the resulting Web page are thus *meaningful* (they form the Web articles that are the main content of the Web site), others are *boilerplate* (they just ensure a common layout of the site, or add contextual information, navigation structure, advertisements, comments). Note that boilerplate may change from one page to another and cannot be assumed to be completely static. In addition, boilerplate can actually take up more volume than meaningful information [16]. Distinguishing between main content and boilerplate is a challenging task [19, 30, 32, 33], with many Web information retrieval and data mining applications: search engines index Web pages based on the informative part of their content; end-users are primarily interested in the main content, and may wish to extract it for readability or accessibility purposes; Web archivists and analysts may wish to archive Web articles independently of the containing Web page [29] to track their evolution irrespectively of changes in layout.

An important design choice of modern Web sites, and a consequence of the use of CMSs, is that Web pages from the same site share a common structure, a structure that can be easily traced in the DOM tree of the Web pages [11]. We further call these pages *sample Web pages*. To illustrate, Figure 1 shows an example of two pages from the same Web site presenting variable content within a fixed template. This structural similarity across Web pages of the same site has been leveraged in a number of information extraction techniques to identify data records from data-intensive, somewhat structured Web sites [1, 7, 24]. A typical use case is the deep Web, where, given sample response pages that result from the submission of a form (e.g., on e-commerce Web sites), the task is to extract all properties (price, name, availability, etc.) of response records.

To identify the information of interest, techniques from the literature have considered the extraction of "informative blocks" [33], pagelets [2, 7], fragments [32] or articles [19, 30] from Web pages. These notions are essentially equivalent: they represent the Web page's *main content*. A variety of techniques has been used in these works: text-based [19, 30], tag-based [36], visual-based [26, 33], or using heuristics on DOM paths [29]. However, in contrast with wrapper induction techniques, these methods operate at the level of a single Web page. Therefore, they ignore an interesting

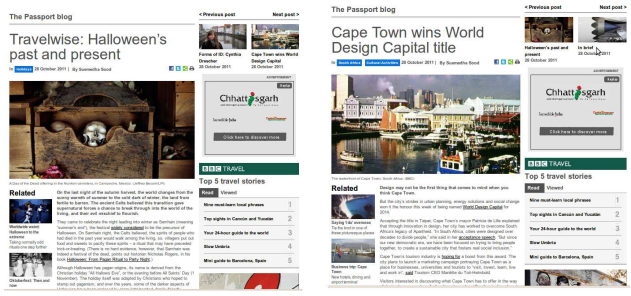

Figure 1: Two sibling news pages (BBC *travel* blog)

feature of pages belonging to the same site: their structural uniformity. We introduce a technique for effective extraction of a Web page's main content, by taking into account both the content itself, and the repeated structure of Web pages that have been generated by the same software resource. The proposed method, called FOREST for "*Focused Object Retrieval by Exploiting Significant Tag paths*", works in a succession of steps:

1. we automatically acquire some keywords for each Web page in the set of sample pages (these keywords may also come from an external source of semantics: query logs, HTML metadata, Web feeds);
2. we identify, at the level of the DOM, structural patterns that are shared by most of the sample pages;
3. we rank these patterns through a novel relevance measure based on information theory and statistics;
4. we infer a generic XPath expression which gives the location of the main content in the sample Web page.

We outline the following contributions of this work:

(i) a novel measure for computing the informativeness of the content of a Web page;

(ii) a technique for wrapper induction at Web site level and automatic identification, using the tag paths of significant DOM elements, of a generic XPath signature of the node that contains the main content of interest;

(iii) effectiveness experiments showing the high accuracy in terms of precision and recall of FOREST over 1,006 Web pages acquired from 93 heterogeneous Web sources, with favorable comparisons, for different settings and baselines including some state-of-the-art methods for article text extraction.

We next discuss the related work before presenting FOREST (Section 3) and the effectiveness experiments (Section 4).

2. RELATED WORK

Relevant content extraction from Web pages is an extensively studied domain [9], of use in many applications such as information extraction, information retrieval, data mining, adaptation of Web pages to small devices, etc.; it is also the spotlight of many online applications[1]. We provide here a brief review of the state-of-the-art.

Automatic Wrapper Induction. Content extraction from Web pages that share structural patterns has often been formalized as a wrapper induction problem [21] for the extraction of data objects, also known as data records [1, 24].

[1]For instance, `http://fivefilters.org/content-only/`

In unsupervised settings, wrapper induction makes use of the common structure of various objects, either at single Web page level [24], or across different Web pages that share the same template [1, 12]. MDR [24] and variants [25, 38] compare string paths that have been identified as being under the same *generalized node* that represents the "data region." A work that compares tag paths as segments rather as strings is [27]. RTDM [12] uses the *tree edit distance* that leverages the common structure of *news* Web pages in order to identify record patterns. However, the template tends to incorporate elements that change from page to page (links, ads, etc.); therefore, changes in content are not reliable enough to decide that content is informative [1, 10]. Also, due to the increasing complexity of HTML pages at DOM level, in the absence of a content relevance measure, these techniques risk detecting structural patterns that have little if no relevance with respect to the extraction target.

Block-based Web Search. In the search for relevant content, segmentation algorithms partition Web pages into "semantic" blocks. Vision-based segmentation is one of the most frequent. VIPS [6] or methods such as [3, 31, 38] have defined a *content relevance measure* using *visual cues*, that is, heuristics on which humans usually rely upon to identify blocks or portions of Web pages that seem to be more interesting than others. However, as argued in [36], given the fast evolving manners of expressing visual properties of text, visual clues are not always reliable and tend to become obsolete with time. Another drawback in the use of visual cues is that the Web page typically needs to be rendered. Therefore, algorithms that use VIPS [4, 26, 37] in the pre-processing phase need large memory resources and processing time. An alternative is to use DOM-based page segmentation. The OMINI [5] system provides an automated way of *learning rules* that helps identify the Web object boundaries. The approach is based on structural features of a tag node (e.g., fanout, size, tag count), combined with heuristics to identify an object separator tag (e.g., repeating tag, standard deviation in the size, etc.).

In order to determine whether a Web page block is *informative*, several different measures have been proposed. For instance, [33] defines a measure of importance for blocks using spatial and content features (e.g., link density or inner text length), but relies on machine learning to identify the best combination of features. We note also [18], which uses an *entropy*-based measure to determine the importance of page blocks.

Exploiting Keywords. A big advantage of using keywords is computational. In contrast to other techniques [27] that consider all tag paths as equally important, analyzing only the interesting ones drastically reduces the computations. The query terms occurring in *search logs* have been proposed as a source of keywords in [8], but with a different goal as ours: to perform an unsupervised structural clustering of Web pages that have been obtained in response to a user query. The access to search logs is however limited either to the Web sites owners or to search engines themselves. In FOREST, keywords are *automatically* discovered, and the technique remains widely accessible.

Using the Web feeds' semantic clues. In our previous work [29], we used Web feeds as clues about the relevant

content in a Web page. The bottom-up algorithm of [29], referred to as SIGFEED in what follows, uses DOM block heuristics to extract the full content of a Web page that is referred to by the respective item in the feed. We have included SIGFEED as a baseline and report the results compared with FOREST in the experimental section.

By pre-processing in FOREST the sample pages into XML documents, and looking for the occurrence of some keywords, we get closer to the rich literature on keyword search over XML documents [17, 34]. Our ranking measure of informativeness applies, not at the level of DOM elements directly, but similarly, at the level of DOM element types as we will see next. However, we significantly differ from these works: first, the XML keyword search works assume keywords as given, while FOREST obtains them automatically; second, the ranking measures are different. In [34] the emphasis is on finding the smallest lowest common ancestor (SLCA) that contains all searched keywords, while we are interested in regions that are dense in keywords but which not necessarily contain all of them. We also diverge in our vision of the ranking measure for XML elements: for instance, [17] uses an adaptation of PageRank. We compare in Section 4 FOREST to a baseline that implements an alternative version of the SLCA algorithm, named COVERAGE.

Text Extraction. A number of recent works [19,30,36] aim at extracting the *textual content* of a Web article in a Web page by relying either on the text or the tag density in subsequent Web page segments. By using a technique used primarily in image processing, similar to [27], CETR [36] computes, per line of HTML code of the Web page, a tag ratio array. The resulting matrix can be fed to a histogram clustering algorithm which filters extraneous data (i.e., boilerplate). BOILERPIPE [19] relies on shallow text features and learning to identify the fulltext of a Web article, and to filter by exclusion the boilerplate present in the respective Web page. Relying only on shallow text features may also conduct to the extraction of other portions of the page which are richer in text than the article itself (for instance, comments). On the other hand, while FOREST relies on sample pages that share the same template, BOILERPIPE works at the level of individual Web pages. We use BOILERPIPE as a baseline method: its extractors have been trained on Web articles, which makes it perfectly applicable to our context.

Template Removal and Change Detection. Template removal methods usually need as input Web pages that share a uniform layout [35]. Tree matching (e.g., the number of occurrences of some branch in the whole forest of DOM trees) or abstract structural features [11] are usually employed as pre-processing before template removal. Most of the time, the problem is reduced to that of finding common DOM subtrees [7] by cross-page clustering for a set of HTML documents.

Tools and Standards. Some tools, such as the Reader mode of the *Safari browser* or similar browser plugins, aim at presenting the main content of a Web page in a visually, more readable way. These tools use a combination of heuristics, site-specific parameters, and estimation of text density.

Several recent development of Web technologies go in the direction of adding more semantics to the markup of a Web page to clearly identify the main content of a Web page. The HTML5 working draft introduces the very useful `<article>` tag to denote a Web page's or a section's main content. HTML5 is not widely used on the Web at the moment, and it is unclear at this point whether the use of such a tag will be consistent enough from a Web site to another. Another initiative is the *hAtom*[2] microformat for syndicated content, which indicates in particular which part of a Web page corresponds to a feed item; the use of *hAtom* on the Web is still marginal, while our approach aims at being generic, without relying on user markup.

3. METHOD

In this section, we describe FOREST's *signifier-aware process* for extracting the main content's block from sample pages.

3.1 Preliminary Notions

Sample pages. Wrapper induction techniques typically work on a set of pages that share a common HTML template. The suitable classes of sample pages are, nevertheless, most of the time manually collected. There are relevant works [8, 11] that structurally cluster Web pages of a Web site. We chose however a simpler, automatic approach to collect structurally similar pages: we use Web feeds.

An increasing number of Web sites incorporate feed facilities so that users can keep in touch with new information. As, first, we put feeds to use for the dataset construction, and, second, we study some feed properties that are relevant to our article extraction problem, we next present some basic information about them.

Web Feeds. *RSS* (acronym for RDF Site Summary, or Really Simple Syndication) and *Atom* are popular feed formats, dialects of XML, that are used to publish frequently updated content such as blog or news entries, audio and video, all in a standardized format. An RSS or Atom document, also called a Web *feed*, contains a *Web channel* that groups various *items* (or entries) that refer to Web articles. In the RSS specification[3], each item has three compulsory elements: title, link, and description. While the *title* gives the name of the Web article (usually) as it appears on the referenced Web page through the *link* URL, the *description* is meant to be a short text describing the respective article. In practice, the description of an item often represents the first lines of the Web article, rather than a real description of its contents.

Signifiers. To spot the main content area on a Web page, and to further determine which DOM elements of the corresponding tag tree are more important than others, we automatically construct a set of relevant terms for that page. As an example of their possible usefulness, we show in Figure 2 how some keywords like "Halloween", "past" and "present" (coming from the title of the respective Web article) are targeting zones in the page. We utilize the notion of *keyword* in the information retrieval sense, as a linguistic clue. However, the high incidence that a keyword may have in the text of a Web article makes it closer to a concept entity (e.g., "Halloween" in our example).

[2] `http://microformats.org/wiki/hatom`
[3] `http://cyber.law.harvard.edu/rss/rss.html`

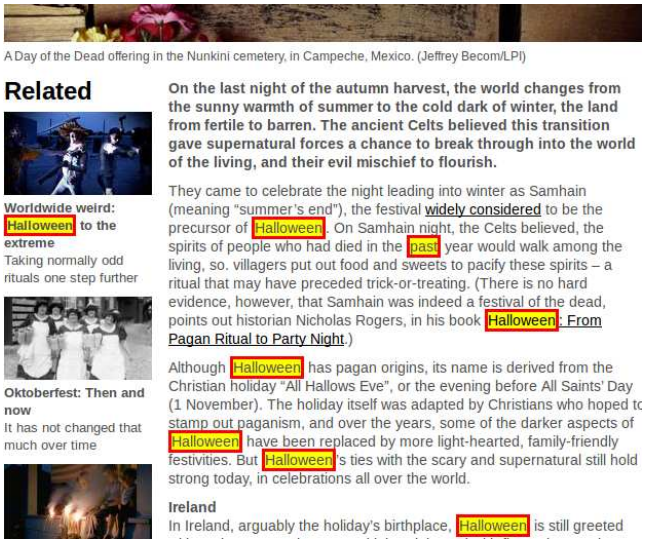

Figure 2: Match of keywords in a Web article

Given a set of sample Web pages, we apply tokenization, stop words removal, and stemming on the text of page. The resulting terms are indexed based on the classic *tf-idf* measure. For each page, the *top-k weighted terms* according to this measure are called *signifiers*. In the experiments, we fix the k threshold to 10 (but discuss other variations). More complex processing, such as POS-tagging or semantic analysis is possible, but not required (preliminary experiments suggest little impact on the results).

We also consider another source of signifiers: the title and description of a feed item. These are also automatically obtained: by a simple parsing of a feed item metadata we get the *title* and a small *description* (typically, first lines or subtitle) of the *Web article object* that we target. This latter method of acquiring signifiers is very straightforward. Without any global analysis on all the textual content of pages, by just extracting terms or n-grams from the *title* and *description* of an item, we obtain some reliable clues about the content that the item pinpoints.

From HTML pages to XML trees. From the Web feeds, we gather the item's URLs and construct a usable version of the HTML page in the form of a DOM tree. We use for this the HTMLCleaner[4] parser, which puts the HTML "tag soup" in the right order and filters out scripts and other tag nodes (e.g., `<noscript>`) that are neither dealt with, nor necessary in our approach. Most importantly, we add to every node in the tag tree a *dfs* attribute that records the order of browsing of a depth-first search walk starting from the root. These *dfs* values serve to identify nodes in the case when they do not have any unique combination of attributes (e.g., `id`, `class`), but also to keep track of possible positions of nodes in the tag tree. This enforced structure is serialized into an *XML document*. We next denote all such documents coming from a single Web feed channel that exposes sample pages d_k, $1 \leqslant k \leqslant r, r \leqslant r$, where n is the number of items in the feed and r represents the number of sample pages that have been annotated.

3.2 Structural Patterns

[4] `http://htmlcleaner.sourceforge.net/`

Significant Nodes. For each analyzed document d_k, we extract all textual leaf nodes that are significant, where a *significant node* is a leaf DOM node whose textual content matches at least one signifier. For precision, leaf nodes that we consider are non-empty textual nodes at the bottom of the DOM tree.

Significant Paths. For each significant node, we construct its tag path. A *tag path* is the sequence of node identifiers from the root to a node in the tag tree. When the node is a leaf, we call this path a *terminal* one.

Definition 1. A *significant terminal path* is a tag path whose last node is significant.

By following the path of signifiers in the DOM, we have an idea about the location of leaf nodes that are significant in a Web page. However, due to the nested structure of elements in the DOM, the location of significant nodes is not sufficient *per se* to identify the boundaries of the article object, which is a more complicated problem [5].

We further analyze only the composing DOM elements of significant paths with the aim of finding structural similarities across different sample pages.

Since the node that generates a significant tag path contains signifiers, the same holds for all its ancestors (i.e., their textual content contains signifiers). The higher in the DOM hierarchy, the more a node tends to contain signifiers; at the same time, it also tends to contain many more non-significant, regular terms. Note that the number of significant paths that have to "decompose" for analysis can be small if the number of signifiers is small, or if the signifiers are consistently pinpointing the same terminal paths.

Element identification. A DOM element typically has a tag name and a list of attributes. However, not all elements have attributes that can make them unique. Paragraphs, table elements, etc., are rarely determined by a unique combination of attributes. Luckily, due to the pre-processing phase, each node has, besides its tag name, at least one *dfs* attribute.

To be able to reuse the element clue from one document to the other, we add a refinement that deals with the fact that, contrarily to what one would expect, `id` and `class` names generated by CMSs may slightly vary from one page to another. In particular, it is common to find, say, a `<div>` tag with a `class` attribute of "post wrapper-09" in one document, and of "post wrapper-02" in another, knowing that empirically they denote the same type of element. We found that a practical way of abstracting out these small differences is to simply keep the first token of the value of an attribute and filter out possible numbers. We next refer to the stemmed value of an attribute as its *tolerant form*. For instance, the tolerant form of the attribute value "post wrapper-02" (occurring in page two) and "post wrapper-09" (occurring in page nine) is simply "post".

Taking the full value of an attribute rather than its tolerant form has however little impact on the results, as we describe in Section 4.

Definition 2. The *element type* of a DOM element is defined as an *XPath expression* constructed based on $\langle t, atts \rangle$, where t is the tag name and *atts* is a set of key-value pairs as follows:

(i) if the DOM element has attributes other than *dfs* (e.g., *id*, *class*) then *atts* is set to the collection of attribute key-value pairs, where the values have been reduced to a tolerant form;
(ii) otherwise, *atts* is set to $\{dfs = d\}$, where d is *dfs* index of the respective element.

The intuition behind an *element type* is that block elements will typically have a way to be referred to in a general manner and grouped across pages (their number of occurence will increase), while, comparatively, more specific elements (e.g., `p`) will be having either a different `dfs` that its homologues from other sample pages, or, comparatively with block nodes will have less signifiers. Examples of simple element types are `//div[@id="container" and @class="post"]` or `//p[@dfs=24]`.

Structural patterns

Definition 3. A *structural pattern* is defined by the combination of an *element type* and the *level* on which the element occurs in the tag tree (i.e., its index in the significant path).

Structural patterns are used to identify similarities between DOM elements across various pages. Indeed, the values of element type and level are typically common to nodes belonging to different sample Web pages (see Figure 3), due to their common generation source (e.g., a script). Figure 3 gives more insights on structural patterns, that have been clustered based on the level on which they occur. Although the elements having the *dfs* equal to 21 and 29 respectively share the same identifying clue, they are counted separately because they reside on different levels.

The *dfs* position of an element is either already present in its *type* or can be added to a structural pattern in order to uniquely identify that DOM element in a particular Web page.

3.3 Informativeness measure

We now introduce the measure of relevance for ranking structural patterns that occur in sample pages, based on the fact they that have been formed by DOM elements that are significant across various documents $d_k, 1 \leqslant k \leqslant r$.

We fix an node element e_i in an XML document d_k. Let x be the number of signifiers in e_i's text, counted with their *multiplicity*. Then all other terms represent non-signifiers, let y be their number. Then $N = x + y$ is the number of terms in the text of e_i. We analogously denote the number of signifiers and non-signifiers in the *whole* d_k in which e_i occurs, that , as X and Y respectively.

Statistical signifier density. One of the most natural ways to determine whether a node is highly significant is to compute its density in signifiers, i.e., $\frac{x}{N}$. However, when N is small, this density might be imprecise, due to lack of observations (a node formed of a single signifier is likely not to be the most significant node of the document). In such contexts, we can use *Jeffrey's add-half rule* [20] as a better statistical estimator of the proportion of signifier terms, yielding $\frac{x+1/2}{N+1}$.

Furthermore, when sampling N elements from a potentially larger set, we have a margin of error on the semantic density. With f the frequency given by the estimator above, the standard deviation is $\sqrt{\frac{f(1-f)}{N}}$ [15]. Assuming 1 standard deviation to obtain a confidence interval of $\approx 70\%$ [15] and combining with the aforementioned estimate, we obtain the following interval for the semantic density of a node:

l	Identifying clue	*dfs*	n
1	`//body`	1	10
2	`//div[@id='container']`	19	10
3	`//div[@class='maincntnr']`	20	10
4	`//div[@class='idem']`	21	10
5	`//div[@class='idem']`	25, 29	10
6	`//div[@class='story']`	78, 82, 83	8
6	`//p[@style='text-align:justify;']`	175	2
7	`//p[@dfs=98]`	98	1
7	`//p[@dfs=107]`	107	1

Figure 3: Partial list of elements occurring in terminal paths for some sample documents, together with their *dfs*, and number of occurrences

$$\frac{x+1/2}{N+1} \pm \frac{1}{N+1}\sqrt{\frac{(x+1/2)\times(y+1/2)}{N}} \qquad (1)$$

We now define as Jeffrey's statistical density, J, the lower value of this interval, i.e., the worst-case estimator at 70% confidence of the semantic density; if this value is less than 0 (because the sample is not large enough), we fix it to 0:

$$J = \max\left(0, \frac{1}{N+1}\left(x + 1/2 - \sqrt{\frac{(x+1/2)\times(y+1/2)}{N}}\right)\right) \qquad (2)$$

As an example, if $x = 8$ and $y = 20$, $J(x, y)$ is comparable to that of a node with $x = 3$ signifiers and $y = 5$ non-signifiers: the lower proportion ($\frac{2}{5}$ compared with $\frac{3}{5}$) is compensated by the smaller number of observations. Even more interestingly, when $x = 1$ and $y = 0$ we have a $J \approx 0.32$, to be compared to the naive density of 1: this element is indeed dense, but due to the low (zero here) number of non-signifiers, we cannot be very sure of its importance. As expected, this density measure tends to favor rather specific nodes, that is, nodes that appear lower in the DOM hierarchy.

Unexpectedness. We derive another approach to significance of a node from the notion of *unexpectedness*, coming from the cognitive model of *simplicity theory* [13] and information theory in general: this measure relies on the observation that humans tend to find a situation interesting when they perceive a discrepancy in complexity.

A situation is unexpected if it is simpler to describe than to generate. Assume a computation model given (say, Turing machine encodings for a given universal Turing machine). Given an object, we consider its generation complexity C_w (i.e., the size of the program that has generated it) and its description complexity C (i.e., its Kolmogorov complexity, the minimum size of a program that describes it); then the unexpectedness of this object is the difference between the two (note that we always have $C \leqslant C_w$). We apply this to the simple setting of non-uniform binomial distributions, that corresponds to our context. Specifically, for each significant node in the DOM tree, we consider its unexpectedness with respect to the number of signifiers and non-signifiers contained in the subtree defined by its location in the DOM. The generation complexity corresponds (up to an additive constant) to the logarithm of the number of

ways to draw $x+y$ elements out of a set of $X+Y$ elements: $C_w = \log(X+Y)^{x+y} = (x+y)\log(X+Y)$. The description complexity, on the other hand, represents the complexity of describing the content of the textual node, knowing that x terms are signifiers: it is the logarithm of the number of ways of choosing exactly x signifiers and y non-signifiers, that is: $C = x\log X + y\log Y$. Finally, the unexpectedness is the difference between these two complexities:

$$U = (x+y)\log(X+Y) - x\log X - y\log Y \quad (3)$$

As a typical example, for a Web page with a total of 20 signifiers and 100 non-signifiers, a node with 10 signifiers and 26 non-signifiers will have an unexpectedness of 23 bits, which is definitely higher than a node with 3 signifiers and 1 non-signifier: 6 bits.

Our preliminary experiments show that unexpectedness favors elements with a large amount of text content that is richer in signifiers than the typical distribution of signifiers on the Web page as a whole. This turns out to be complementary to the statistical density J, which favors nodes poor in non-signifiers.

Informativeness.

$$I(sp_i, d_k) = J(sp_i, d_k) \times U(sp_i, d_k) \quad (4)$$

This measure characterize the informativeness of a structural pattern $sp_i, i \in 1:m$, where m is the total number of structural patterns that are shared by our sample pages, as the product between the unexpectedness and the statistical signifier density of a DOM element that has the structural pattern sp_i in document d_k.

3.4 Combining structure and content

A global measure of relevance for a structural pattern combines the informativeness of it (Section 3.3) with its number of occurrences and a decay factor given by its level (Section 3.2). In terms of the number p of significant terminal paths where sp_i occurs on *level*, we have:

$$R^{\text{FOREST}}[sp_i] = \sum_{k=0}^{p} I(sp_i, d_k) \times p \times level(sp_i) \quad (5)$$

There exists a single sp_i (element type on a certain level), so the number of occurrences of sp_i in the sample pages is p, $p \leq n$, where n is the total number of documents. The role of the p factor is clear, since we want to give a bigger weight to structural patterns that are not only informative, but also very frequent. In addition, the *level* factor is a heuristic favoring nodes that are deeper in the DOM tree. Indeed, elements that are too high in the hierarchy (e.g, `<body>`) are more unlikely to effectively identify the target article object because they are not discriminative enough. The idea of a decay factor has been also introduced in other works [17, 22], under different forms. For instance, in the ranking formula of [17] the decay factor is a value in the range 0 to 1.

We rank the structural patterns $sp_i, i \in 1:m$ using this relevance measure. In the end, we simply derive the XPath clue of the best ranked structural pattern, which at this point fully identifies a target subtree across various documents. For clarity, following the reasoning on Figure 3, the path of the generic wrapper will be: `//div[contains(@class,'story') and (@dfs='78' or @dfs='82' or @dfs='83')]`. By applying the generic element clue as a XPath expression over the $d_k, k \in 1:r$, we are most likely to find a node element that satisfies these structural conditions and whose content is highly informative.

We summarize the final part of FOREST in Algorithm 1 (for space reasons, we assume clear the pre-processing, concepts definition and we use the measures previously described):

Input: *elementTypes*
Output: *infoBlock* : the most informative block of a Web page
foreach *possible level* **do**
  get the *structuralPatterns* occuring on this *level*;
  foreach *elementType* of *structuralPattern* **do**
    count the *nbofOccurrences* of *elementType*;
    possibleDfs := group all its *dfs* positions across *xmlDocuments*;
  end
  fullXPathClue := *elementType* completed with its *possibleDfs*;
  compute the *relevance* of *structuralPattern* ;
  construct a *genericWrapper* := $\langle fullXPathClue, relevance\rangle$;
  candidates.add(*genericWrapper*):
end
sort *candidates* based on their *relevance*;
return *infoBlock* := the top ranked element from *candidates*;

Algorithm 1: Ranking structural patterns

3.5 Coverage

Intuitively, the DOM node which contains the main content of an article should be defined as the smallest, lowest common ancestor (SLCA [34]) node in the hierarchy that has a maximal coverage. For a structural pattern $sp_i, i \in 1:m$, the *coverage* of it represents the sum of normalized *tf-idf* weights of signifiers occurring in the text of a DOM element that has sp_i in d_k.

$$Cov(sp_i, d_k) = \frac{\sum_{j=0}^{nbSigs(node_{(sp_i,d_k)})} weight(signifier_j)}{totalNbOfSignifiers} \quad (6)$$

We implement this as a baseline to show what can be achieved by making use of the bag of signifiers, in comparison with the more sophisticated measures used by FOREST. For this setting, COVERAGE uses $Cov(sp_i, d_k)$ to replace the informativeness measure in Formula 7, and selects a structural pattern that is best covered in terms of signifiers:

$$R^{\text{COVERAGE}}[sp_i] = \sum_{k=0}^{p} Cov(sp_i, d_k) \times p \times level(sp_i) \quad (7)$$

4. EXPERIMENTS

Dataset construction. Next, we describe the *RED* (for *RSS-based Experimental Dataset*) dataset used to evaluate all techniques discussed in this section. Note that the existence of Web feeds is not a condition for FOREST, which only needs some sample pages that may represent *per se* the source of keywords. We have used Web feeds not only as a potential, alternative source of keywords, but also because the feed items of a Web channel refer to Web pages that typically share the same template.

The motivation for construction of this dataset is the current impossibility to test FOREST directly on existing Web article content extraction datasets: first, they operate at Web page level [19]; second, the datasets that are using the setting of sample pages for the main article extraction are not online [12]; finally, datasets using sample pages may exist, but are used instead in the context of the deep Web for response record extraction. In addition, one of the source of signifiers that is common to FOREST, but also to the SIGFEED baseline is Web feeds, and there exists, at our best knowledge, no other feed-based dataset for Web article content extraction.

Feeds of Web sites are acquired in an automatic manner by scraping the results of a *feed meta-search engine*, Search4RSS[5]. The condition of selection of feeds is to be parseable and to point to Web articles (and not tweets, for instance). Both feed and reference Web pages have been crawled at a given point in time, for 90 Web sites, with 3 exposing two slightly different templates. We have thus accumulated 93 types in total and 1,010 sample Web pages. Note that the annotation process is particularly time-consuming since more than 1,000 Web pages need to be annotated by hand.

As mentioned, feed URLs were given by Search4RSS in response to a topic query (keyword-based). For this reason, *RED* is quite heterogeneous: it includes various types of Web articles that exists on the Web on a particular subject (e.g., poetry), such as blog posts, news pages, professional or personal Web pages, etc. There exist various particularities of Web articles that we have observed during the annotation phase; for instance, we have found main article content scripted, spreading across different pages, or mainly composed of images or videos. Content is represented in new ways, which increases the difficulty of the extraction task. The gold standard is further discussed and available at the first author's Web page.[6]

Gold standard. Remember that the target of the extraction is a Web article, so the goal is to retrieve the *title*, *fulltext*, and metadata like author(s), publication date, image captions, categories and tags, if they exist.

The gold standard for our dataset has been manually annotated. We have also annotated multiple, random Web pages corresponding to feed items. This is useful in the analysis of the number of pages that are necessary to reach a top efficiency. On the other hand, not all Web feeds have the same number of items in their channel, so we have annotated between 2 and 20 Web pages per feed, knowing that the typical number of items in a feed is 10 [29]. After a round of quality assurance to check that the guidelines were well understood, the annotation task is intuitive enough to reach a high-level of inter-annotator agreement; the precision from one annotator to another was 97%.

Baselines. We compare the two variants of FOREST (when keywords come from feeds, or from the tf-idf analysis) to four different baselines.

The first is BOILERPIPE [19], already introduced in Section 2. As a state-of-the-art method in content extraction, BOILERPIPE uses quantitative linguistics (features like average word length, absolute number of words, and the like) mingled with some heuristics on the DOM tree and semi-supervised learning to identify fragmented, short text in blocks of a document as *boilerplate*, and filter it in order to obtain the main article content. Unlike FOREST, BOILERPIPE needs to be trained for specific data, but a pre-trained extractor (i.e., *ArticleExtractor*) that we believe to be the best adapted for the articles in *RED*, is publicly available in the author's implementation.[7] No significant differences were observed for other provided extractors.

Another baseline, that we have previously developed, is SIGFEED [29]. This technique selects, at the level of a single Web page (similarly to BOILERPIPE), the smallest, lowest `<div>` block ancestor in the DOM hierarchy that is the most dense in keywords obtained from the feed description (similarly to FOREST (feeds)).

The COVERAGE (see (6)) baseline is one intuitive technique that takes into account the tf-idf weighted signifiers that we automatically acquire in the presence of multiple sample pages. This heuristic is useful to test whether our elaborate informativeness measure adds value.

Finally, the DESCRIPTION heuristic simply takes, as the main content of the Web page, the title and description of an item as it appears in the Web feed (with some processing on the description to eliminate the possible HTML encoding). This hypothesis is important to be tested in the case of Web feeds, because there are cases in which feeds contain the whole title and fulltext of an article.

Performance metrics. The result of our technique is a *generic tag path* which returns, for each Web page of studied channel, a DOM subtree in the form of an *XML document*. This is useful to get any media resource that is typically incorporated in an article which contributes to its object view. In spite of that, we make the evaluation on the extracted textual content, to compare it with our baselines (in particular, because the output of BOILERPIPE and DESCRIPTION is plain text). We also want the comparison measure to be insensitive to different amounts of whitespace extracted by various methods. After a typical normalization, we compute the set S of 2-grams (two consecutive words) in the output of all methods, and estimate classical *precision* and *recall* measures by comparing it to the set G of 2-grams of the gold standard, as:

$$\text{Precision}(G,S) = \frac{|G \cap S|}{|S|}, \qquad \text{Recall}(G,S) = \frac{|G \cap S|}{|G|}.$$

Precision and recall are then summarized by their harmonic mean, the F_1 measure. Note that the precision we compute is exactly the ROUGE-N [23] measure used for comparing the performance of text summarization techniques.

Main results. We show in Table 1 the mean precision, mean recall, and corresponding F_1 measure of the different methods tested over the whole dataset. We note that, since we have a sample of 90 independent sites and values of the order of 90%, the confidence interval at 95% probability (1.96 standard deviation) [15] is ± 0.06. To investigate more precisely the shape of the distribution of results for each method, Figure 4 presents the F_1 measure of the different methods. We show for each method its 9th and 91th percentile (whiskers), its

[5] `http://www.search4rss.com/`

[6] `http://perso.telecom-paristech.fr/~oita/research.html`

[7] `http://code.google.com/p/Boilerpipe/`

	Prec. (%)	Rec. (%)	F_1 (%)
Forest (tf-idf)	**93**	**92.8**	**92.1**
Forest (feeds)	**92**	**90.5**	**89.6**
Boilerpipe	79.5	84.0	81.7
Sigfeed	88	83.9	84
Coverage	89.7	83	82.9
Description	84.4	22	30.3

Table 1: Mean precision, recall, and corresponding F_1-measure

first and third quartile (box) and its median (horizontal rule).

Both variants of Forest significantly outperform the baselines, with a global F_1 measure of, respectively 92.1% and 89.6%. These results were obtained for the whole dataset, that is, 1006 Web pages.

Boilerpipe achieves a relatively low score here, despite the fact that the Web pages of our dataset (blog posts, news articles) match the kind of Web pages the *ArticleExtractor* has been trained on. We observe in practice that Boilerpipe has the following shortcomings: when a Web page contains various small Web articles, the text of all is taken as a whole. Also, when the text of an article is segmented by the use of images (or different kind of content that text), Boilerpipe considers them as separators, giving in this case a partial result. At the same time, Boilerpipe can be applied directly at the level of a single Web page, which is a more independent setting than that of Forest, in which at least two pages that share the same template are needed.

The intuitive Coverage approach that uses a relevance measure based on weighted keywords and the Sigfeed [29] heuristic-based method manages a higher level of F_1 measure than Boilerpipe. We infer from this that, wherever their source, keywords are globally useful in the task of content extraction. Sigfeed is however outperformed by Forest: the simple `div` block heuristic that works in many cases, fails to fully extract the article when complex HTML element nesting is involved.

Precision of the Description baseline is low, suggesting first, that feed items also contain 2-grams that do not appear in the main content (an example of that are dedicated links to go to the unabridged version), and second, judging by the abysmal recall, that feed items are often incomplete versions of the main content of a Web page. We also found out that, for practical (the article can be very long) or commercial purposes (to attract site visitors), many feed generators just cut the description to a couple of lines [14].

To look more carefully into these results, turn now to Figure 4. This graph shows in particular that in addition of having better performance in average, the two variants of Forest are also more robust: on 90% of the corpus (resp., 75%), the F_1 measure is greater than 84% (resp., 91%), to compare with 55% and 73% for Boilerpipe.

Another interesting feature shown in Figure 4 is that both Sigfeed and Coverage have quite a high median, which means they will work well on most sources, but have a F_1-measure less than, respectively, 55% and 9%, on 10% of the corpus. As already noted, Description performs very poorly, with a F_1 score greater than 50% on less than 25% of the corpus.

The similar performance of Forest(tf-idf) and Forest(feeds) suggests that keywords extracted from Web sites themselves are as useful as keywords from more trusted sources like Web

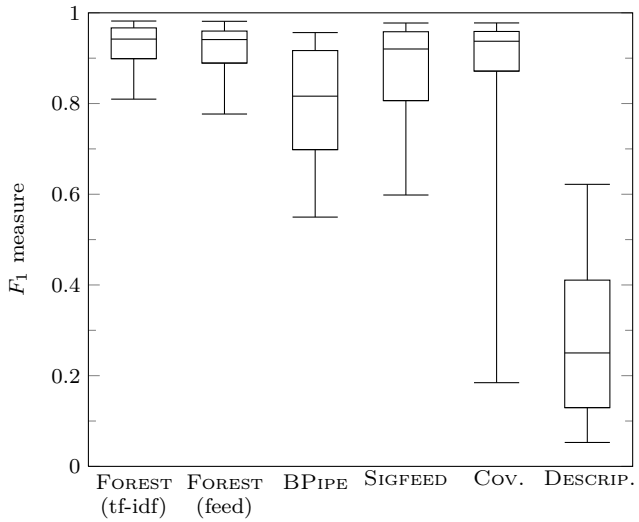

Figure 4: Box chart of the F_1 measure

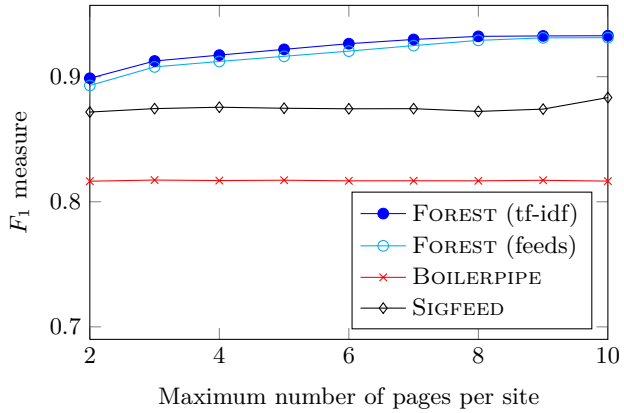

Figure 5: Evolution of the F_1 measure in function of the (maximum) number of pages sharing a similar layout considered for each site

feeds items. However, Forest(tf-idf) has the advantage of not depending on the presence of Web feeds.

Influence of the number of pages. To understand the impact of the number of pages with the same layout available to Forest, we plot in Figure 5 the obtained F_1 measure of different methods with respect to the number of pages sharing the same template. Obviously, since neither Sigfeed nor Boilerpipe make use of the repeated structure, the variation of their F_1 measure here is not significant: it is just due to the somewhat fluctuating performance behavior on the collection of Web pages of a given site.

Forest(feeds) is already at the same effectiveness level as the Sigfeed baseline that does a comparable job, and is even slightly better, perhaps thanks to the measure of informativeness used.

Forest requires at least two pages sharing the same layout: this is helpful not only for the acquisition of discriminative keywords using *Tf-Idf*, but also allows the exploitation of the repeated Web page structure for pattern identification. As soon as there are at least two sample pages, Forest reaches an F_1 score that is already above that of Boilerpipe.

When the signifiers are given by a *Tf-Idf* analysis on the

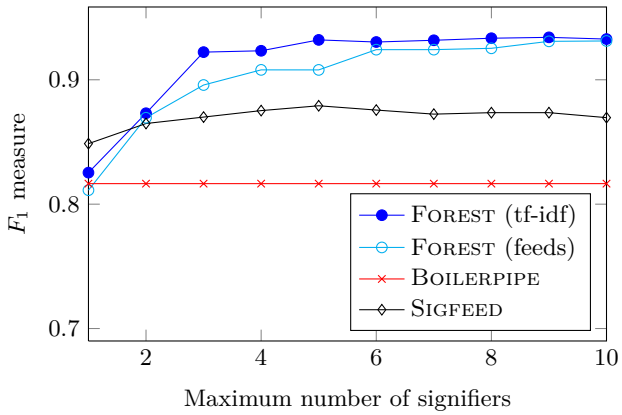

Figure 6: Evolution of the F_1 measure in function of the (maximum) number of signifiers used in the process of path selection

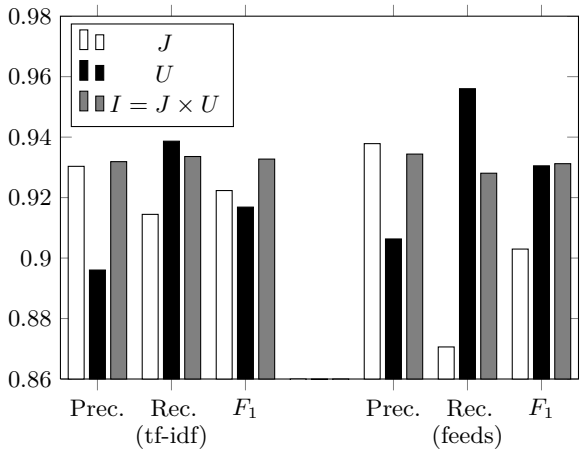

Figure 7: Influence of J and U on mean precision, recall, and corresponding F_1-measure of Forest

Web pages themselves, Forest cannot be applied with good precision on a single page: this is expected, since it means that the IDF measure is here constant and cannot serve to distinguish between page-specific and terms which are common to the Web site as a whole. In addition, there is no repeated structure that can serve to add relevance to tag paths. Forest keeps improving as the number of Web pages increases, to reach a plateau around 8–10 pages. In any case, a small number of pages is enough to get better results than the baselines, which broadens the applicability of the method.

Influence of the number of keywords. Another parameter that can be modified is the number of signifiers kept for a given Web page. From our experiments (see Figure 6), as long as the number of signifiers exceeds 5, the quality of the extraction is not overly affected, though we do observe a slight reduction in effectiveness when too many signifiers are considered.

When the number of signifiers falls below 5, the resulting few terminal paths are not giving enough insight on the informativeness of their element patterns, and the precision of results gets lower.

Boilerpipe does not uses signifiers, so the variation of its F_1 measure is zero. On the contrary, Sigfeed does, and we observe a surprisingly high F_1 stability, with a high score, even when using a single keyword. This could be explained by the `div` heuristic that is employed in Sigfeed: the idea that significant nodes are simply clustered based on their first (i.e., lowest in the hierarchy) `div` ancestor. Obtaining low variations of the efficiency when less signifiers are used to get significant nodes could mean either that these significant nodes have in common the same `div` block. As the sources of signifiers for Sigfeed are the title and description of an item, the feed signifiers tend to concentrate at the beginning of the Web article, at the same relative location. So having many other keywords in this case does not change the lowest common block ancestor that is chosen as wrapper for the target article.

Miscellaneous. We report briefly on additional variations of the settings for Forest.

We have tested U and J separately in the beginning to find a suitable measure (see Figure 7). The common pattern is that, both for Forest (tf-idf) and Forest (feeds), J and U alone give reasonable F_1-measure scores, though lower than the combination of $J \times U$. In particular, J tends to have a higher precision than U and a lower recall, which makes the combination of the two a good compromise.

Challenges. Even if the DOM block is correctly identified, Forest efficiency can be lowered in some cases by the fact the extraction may also contain comments or related links. The cause is simple: the signifiers can also pinpoint comments or links; when comments or links are integrated together with the main content in a DOM block without a proper logic segmentation, the common block is taken as result.

We have annotated the gold standard regardless of the actual relevance that comments or related links may have to signifiers, with the aim to extract only the main content. It is however difficult to decide whether this type of related content can not be useful for a user that issues a keyword query.

We have made experiments to filter out first, the lists of anchors from the DOM of the document in the pre-processing phase, and second, significant paths that have the keyword "comments" in their signature. However, these heuristics barely improve the results: 1% for the first and 2% for the latter. The reason that we observe in practice is that either the lists of anchors is part of the gold standard for some articles, or the heuristics do not work because the anchors or comments are encoded in multiple, subsequent `div`s rather than in list items in `ol, ul`, etc typical DOM lists types. The improvement is also minimal for other heuristic choices like the use of the tolerant form of a DOM node attribute in definition 2 (1%), and biasing Forest in favor of deep nodes by a decay level factor in relevance formula (7) (2%).

5. DISCUSSION

We have presented a novel unsupervised technique that mingles wrapper induction with content analysis, for Web pages that share the same HTML template. The algorithm has the originality of using keywords to trace locations of interesting blocks in the Web pages. We filter out significant tag paths and define a measure of relevance at the level of DOM elements. This measure takes into account not only the structural patterns of the elements, but also the content-

based informativeness. This approach produces a single, generic tag path that is used to extract the data of interest across the various pages. FOREST achieves promising results in comparison with state-of-the-art approaches for content extraction, for a diversified dataset of Web pages containing Web articles.

In this work, we have shown that we can successfully exploit two potential sources of keywords: Web feed items and (if enough pages using the same layout are given) frequent and informative terms occurring on each Web page. We believe that the same technique can be used on a broader range of settings: other possible sources of keywords are anchor text of links pointing to a page, terms occurring in search engine query logs, semantic metadata associated to the Web page. We have also successfully applied FOREST to the extraction of data from deep Web response pages generated from submitting a form [28]: here, the signifiers can come from the keywords used during the form submission.

6. REFERENCES

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *Proc. SIGMOD*, June 2003.

[2] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proc. WWW*, May 2002.

[3] P. Bohunsky and W. Gatterbauer. Visual structure-based Web page clustering and retrieval. In *Proc. WWW*, Apr. 2010.

[4] E. Bruno, N. Faessel, H. Glotin, J. L. Maitre, and M. Scholl. Indexing and querying segmented web pages: the block Web model. *World Wide Web*, 14(5-6), 2011.

[5] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the World Wide Web. In *Proc. Distributed Computing Systems*, Apr. 2001.

[6] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. VIPS: a vision-based page segmentation algorithm. Technical Report MSR-TR-2003-79, Microsoft, 2003.

[7] J. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: focused extraction of QA-pagelets from the deep Web. In *Proc. ICDE*, 2004.

[8] D. Chakrabarti and R. R. Mehta. The paths more taken: Matching DOM trees to search logs for accurate Web page clustering. In *Proc. WWW*, Apr. 2010.

[9] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of Web information extraction systems. *IEEE TKDE*, 18(10), 2006.

[10] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *Proc. VLDB*, Sept. 2001.

[11] V. Crescenzi, P. Merialdo, and P. Missier. Clustering Web pages based on their structure. *Data and Knowl. Eng.*, 54(3), 2005.

[12] D. de Castro Reis, P. B. Golgher, A. S. da Silva, and A. H. F. Laender. Automatic Web news extraction using tree edit distance. In *Proc. WWW*, May 2004.

[13] A. Dimulescu and J.-L. Dessalles. Understanding narrative interest: Some evidence on the role of unexpectedness. In *Proc. CogSci*, July 2009.

[14] E. Finkelstein. *Syndicating Web Sites with RSS Feeds for Dummies*. Wiley Publishing, Inc., 2005.

[15] D. Freedman, R. Pisani, and R. Purves. *Statistics*. W. W. Norton, Jan. 1998.

[16] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *Proc. WWW*, 2005.

[17] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *Proc. SIGMOD*, 2003.

[18] H.-Y. Kao, J.-M. Ho, and M.-S. Chen. WISDOM: Web intrapage informative structure mining based on document object model. *IEEE TKDE*, 17(5), 2005.

[19] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proc. WSDM*, Feb. 2010.

[20] R. E. Krichevsky and V. K. Trofimov. The performance of universal encoding. *IEEE ToIT*, 27(2), 1981.

[21] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proc. IJCAI*, Aug. 1997.

[22] S.-J. Lim and Y.-K. Ng. An automated change-detection algorithm for HTML documents based on semantic hierarchies. In *Proc. ICDE*, Apr. 2001.

[23] C.-Y. Lin. ROUGE: a package for automatic evaluation of summaries. In *Proc. Workshop on Text Summarization Branches Out (WAS)*, July 2004.

[24] B. Liu, R. L. Grossman, and Y. Zhai. Mining data records in Web pages. In *Proc. KDD*, Aug. 2003.

[25] B. Liu and Y. Zhai. Net - a system for extracting Web data from flat and nested data records. In *Proc. WISE*, 2005.

[26] R. R. Mehta, P. Mitra, and H. Karnick. Extracting semantic structure of web documents using content and visual information. In *Proc. WWW*, May 2005.

[27] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the Web using tag path clustering. In *Proc. WWW*, Apr. 2009.

[28] M. Oita, A. Amarilli, and P. Senellart. Cross-fertilizing deep Web analysis and ontology enrichment. In *Proc. VLDS*, Istanbul, Turkey, Aug. 2012. Vision article.

[29] M. Oita and P. Senellart. Archiving data objects using Web feeds. In *Proc. IWAW*, Oct. 2010.

[30] J. Pasternack and D. Roth. Extracting article text from the Web with maximum subsequence segmentation. In *Proc. WWW*, Apr. 2009.

[31] Z. Pehlivan, M. Ben Saad, and S. Gançarski. A novel Web archiving approach based on visual pages analysis. In *Proc. IWAW*, Sept. 2009.

[32] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis. Automatic detection of fragments in dynamically generated Web pages. In *Proc. WWW*, May 2004.

[33] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for Web pages. In *Proc. WWW*, May 2004.

[34] C. Sun, C.-Y. Chan, and A. K. Goenka. Multiway SLCA-based keyword search in XML data. In *Proc. WWW*, 2007.

[35] K. Vieira, A. S. da Silva, and N. Pinto. A fast and robust method for web page template detection and removal. In *Proc. CIKM*, Nov. 2006.

[36] T. Weninger, W. H. Hsu, and J. Han. Content extraction via tag ratios. In *Proc. WWW*, Apr. 2010.

[37] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in Web information retrieval

using Web page segmentation. In *Proc. WWW*, May 2003.

[38] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proc. WWW*, May 2005.