Automatic Wrapper Induction from Hidden-Web Sources with Domain Knowledge

Pierre Senellart
INRIA Saclay
& TELECOM ParisTech
Paris, France
pierre@senellart.com

Avin Mittal
Indian Institute of Technology
Bombay, India
avin@cse.iitb.ac.in

Daniel Muschick
Technische Universität Graz
Graz, Austria
daniel.muschick@gmail.com

Rémi Gilleron
Univ. Lille 3 & INRIA Lille
Villeneuve d'Ascq, France
remi.gilleron@univ-lille3.fr

Marc Tommasi
Univ. Lille 3 & INRIA Lille
Villeneuve d'Ascq, France
marc.tommasi@univ-lille3.fr

ABSTRACT

We present an original approach to the automatic induction of wrappers for sources of the hidden Web that does not need any human supervision. Our approach only needs domain knowledge expressed as a set of concept names and concept instances. There are two parts in extracting valuable data from hidden-Web sources: understanding the structure of a given HTML form and relating its fields to concepts of the domain, and understanding how resulting records are represented in an HTML result page. For the former problem, we use a combination of heuristics and of probing with domain instances; for the latter, we use a supervised machine learning technique adapted to tree-like information on an automatic, imperfect, and imprecise, annotation using the domain knowledge. We show experiments that demonstrate the validity and potential of the approach.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Design, Experimentation

Keywords

Hidden Web, deep Web, invisible Web, information extraction, probing, wrapper, form, Web service

1. INTRODUCTION

Access to Web information today primarily relies on keyword search engines. These search engines deal with the *surface Web*, the set of Web pages directly accessible through hyperlinks, mostly ignoring the vast amount of information hidden behind forms, that composes the *hidden Web* (also known as *deep Web* or *invisible Web*). This information, often of high quality (think, for instance, of all kinds of *Yellow Pages* directories), cannot easily be dealt with in an automatic way. A 2001 study [2] estimated that dozens of thousands of such hidden-Web sources existed (this order of magnitude was raised to hundreds of thousands in a more recent work [10]). All such *services* of the hidden Web have a different interface—while some are machine-friendly Web services, most use an HTML form interface with HTML result pages—and it is a time-intensive task to write *wrappers* for each. While supervised machine learning approaches [8, 18] exist for simplifying this wrapper construction step, in this paper we aim at a fully automatic, unsupervised, approach for wrapping a service of the hidden Web.

To restrict the scope of this particularly broad problem, we limit ourselves to services related to a given *domain of interest*, described by *domain knowledge*. Clearly, with human providing feedback, supervised techniques can go further toward a better understanding of the Web. But the kind of unsupervised approach we propose is (i) useful at least as a first step, before any human intervention; (ii) often the only one available for applications when human resources cannot be used; (iii) essential if we want the system to adapt to the scale and diversity of the Web, as well as its dynamism.

Consider a service of the hidden Web, say an HTML form, that is relevant to the particular application domain. To understand its semantics, a first step is the analysis of the form, i.e., the structure of the service inputs. We use some heuristics to associate domain concepts to form fields, and then *probe* these fields with domain instances to confirm or infirm these guesses. We then use clustering techniques for distinguishing between *result pages* and *error pages*. The next step is to extract information from the results of the form, i.e., HTML pages. Result pages are annotated with domain concepts (based on available domain instances that are recognized). This annotation, which is both imprecise and incomplete, is used by supervised information extraction techniques adapted for tree-structured HTML pages, i.e., *conditional random fields for trees*.

We first introduce the type of domain knowledge that we consider in Section 2. Then we successively present sys-

WIDM'08, October 30, 2008, Napa Valley, California, USA.
Copyright 2008 ACM 978-1-60558-260-3/08/10 ...$5.00.

tems that probe a form in order to understand its input (Section 3) and that extract information from result pages (Section 4). We next show how both systems can be used together in order to wrap a given source of the hidden Web as a regular Web service in Section 5. We finally present experimental results on real hidden-Web sources in Section 6.

Before going into the details, let us highlight the main contributions of this paper with respect to the related work discussed in Section 7:

1. A system for wrapping a source of the hidden Web as a Web service in an automatic way, using domain knowledge given in a predefined form, to allow for domain independence of the approach.
2. A simple yet effective way of clustering result and error pages, based on terminal paths in the DOM tree.
3. An original application of a supervised machine learning technique in an unsupervised context, with the help of an incomplete and imprecise annotation with domain knowledge.
4. Experiments that show the viability and potential of the approach.

More detailed information about the system described in this paper can be found in [16, 17, 24] (unpublished works).

2. DOMAIN KNOWLEDGE

As understanding services of the hidden Web is a very broad, difficult, and undoubtedly AI-complete problem, we limit our interest to a specific *application domain*, relying on available *domain-specific knowledge*. This will be illustrated in this paper by the domain of research publication databases, but the approach proposed here is quite general and can be applied to any domain, provided that the necessary knowledge described here is given. The needed domain knowledge can be decomposed into two different parts: *domain concepts* and *domain instances*.

Domain concepts. The domain concepts are just a set of *concept names* that describe the different concepts that may appear in this specific domain. We shall, for example, consider the following self-describing concepts for the research publication domain: `Title`, `Author`, `Date`, `Journal`, `Conference`. Additional concepts (e.g., `Publisher`, `Pages`, `Volume`) could be added in a similar way. Knowledge about each concept is given next as domain instances.

Domain instances. We deal here with concrete representations (as strings) of concepts. Observe for instance that the strings *June 2007* and *07/06* may both stand for the same instance of a `Date` concept. Domain instances are character strings that stand for instances of the domain concepts. The following steps are then carried out. First, a representative set of words appearing in these strings, with their corresponding frequency, is selected as an approximate frequency distribution of domain instances. Second, for each given string s, if s may stand for an instance of concepts $c_1 \dots c_n$, we assign $1/n$ as the (approximate) probability that s stands for an instance of each concept c_i.

Note that acquiring the domain knowledge is done only during initialization and once for each domain. The system is fully unsupervised once the domain knowledge has been built, and the only other external input that we use in the following sections is a general ontology for the language that we consider (Wordnet in this case). So, the system can be quite easily generalized to services in other languages by simply changing the ontology, and to other domains by using the knowledge from some other domain.

Let us describe the domain instances that we use in the case of the research publication domain. We downloaded the content of the DBLP database as an XML file from `http://dblp.uni-trier.de/xml/`, and we used it to generate our domain instances. For the concepts `Title`, `Journal`, `Conference`, we used the basic technique described above. For the `Date` concept, we provide a specific entity recognizer (in the form of a set of regular expressions describing monthly or yearly dates). For the `Author` concept, we extracted first and last names from DBLP person names with some heuristics, and use regular expressions describing the ways to recombine first names and last names (forming for instance *Smith*, *Smith, John*, *John Smith*, *Smith John*, *Smith John Jack* with various probabilities, from the last name *Smith* and the first names *John* and *Jack*).

For the last two cases, probabilities associated with a regular expression are chosen in a quite *ad hoc* way. Ideally, they should come from statistical evaluation on large corpora. Note that the use of the DBLP data results in a quite computer-science–centric domain knowledge, but this may not necessarily be a problem for dealing with general research publication databases, as discussed in Section 6.

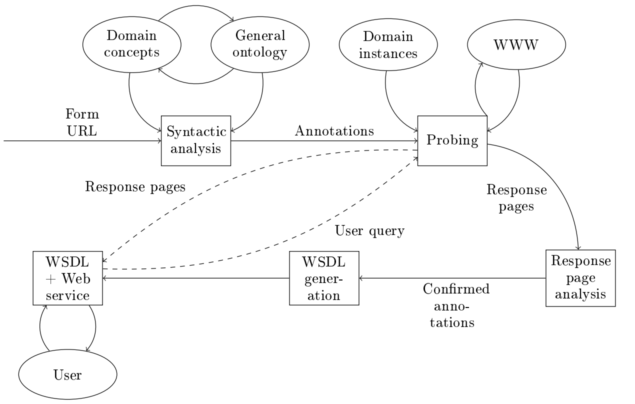

Figure 1: Architecture of a system for probing the hidden Web

3. PROBING THE HIDDEN WEB

Most hidden-Web services are only accessible through an HTML form interface, and their results are shown as HTML result pages. The first step in understanding and indexing these services is to understand the structure of the form interfaces and their result pages. As discussed in Section 2, we rely on some domain knowledge to analyze the structure of HTML forms; in particular, we use here as domain knowledge (i) a list of domain concepts, and (ii) words appearing in instances of each concept.

Given the URL of an HTML form, the aim of the work presented in this section is to annotate each relevant field of the form with the domain concept it maps to, to confirm these annotations by probing the form with domain instances (that is, submitting the form with some filled-in field), and to retrieve the result pages that are analyzed in the process described in the subsequent section. We describe the system using the *scientific publications* domain, but our approach is quite general and can be applied to any other domain for which we have appropriate domain knowledge.

We make some simplifying assumptions on the structure and semantics of the forms. First, we assume that forms support *partial match* queries. For instance, a field for typing in the title of an article may be queried with a single word from the title; we never found any form where this assumption failed to hold in our experiments. *Second*, we assume that there are no specifically required fields in the form, and that each field can be either filled-in or omitted. This is actually quite a strong assumption, and it would be a very interesting extension to use probes to distinguish between required and optional fields.

Our system for probing the hidden Web is shown in Figure 1. The different modules, which are described next, are shown as rectangular boxes, while external data and agents are represented as ellipses. Four different modules constitute the main part of the system, while a fifth one is used to handle Web-service requests from the user. The *syntactic analyzer* processes the HTML code of the form, extracts relevant information, and adds initial annotations. The *probing* and *response page analyzer* modules probe the form with domain instances, so as to confirm or refute these annotations. Finally, the resulting analyzed form is wrapped as a Web service (*WSDL generator*) that can be used to provide a user with an abstract interface to a given form of the hidden Web. We describe these modules in the following sections.

Structural Analysis of an HTML Form. The role of the first module of our probing system is to analyze the structure of the form and to find fields that are relevant to the domain concepts. The proper (semantic) way to give the label of a form field is the use of the HTML element `label`, whose `for` attribute must correspond to the `id` attribute of a form field [28]. Unfortunately, this tag is rarely used by Web developers and we thus have to resort to other contextual information, such as the `name` or `id` attribute, or the text appearing before the field (in the source code). An alternative is to use the graphical layout of the form, as in [22, 32], and rules indicating where the label of a field most often lies relative to the field.

```
Algorithm 1 Structural analysis of an HTML form
Input: URL of a form.
Output: List of fields with probabilistic annotations.
(a) Retrieve the Web page at the given URL.
(b) For each form field:
      (i) Gather all words of the textual context: name and
          id attributes, content of the corresponding label
          element, words appearing before the field.
     (ii) Remove stop-words and stem all context words
          with Porter's stemming algorithm [20].
    (iii) Check whether any resulting stem matches a stem
          of concept related words as given by WordNet [21].
     (iv) Annotate the fields with matching concepts, with
          the associated confidence as the probability that
          this field represents this concept.
```

We present in Algorithm 1 our algorithm for structural analysis of a form. Once contextual information is retrieved for each field, some standard preprocessing (stop-word removal, stemming) is applied to words of the context, before comparison to words related to concept names. Related words are extracted from WordNet [21] (they could also come from a domain-specific ontology) by following relations such as synonymy, meronymy, etc. Matches between the context of a field and words related to a concept correspond to an annotation of the field with the concept name, subject to a confidence that is computed from the source of context words and the distance between matched and concept words in the ontology graph. Confidence values are at the moment chosen a bit arbitrarily. Assigning useful (probabilistic or statistical) interpretations to these confidence values and making further use of them for interpreting the probing results is an interesting direction and should be addressed in future work.

Form Probing. Probabilistic annotations of concepts are confirmed by probing the form. Specifically, we compare what happens when we probe a field, annotated (automatically, as described earlier) as concept c, with instances of this concept, chosen representatively of the frequency distribution of instances of c in the domain database. If the result pages we obtain are significantly different from result pages obtained by probing the field with nonsense words (e.g., `dsqkhzei`), we may conclude that the annotation is indeed correct. Algorithm 2 describes this confirmation step.

One of the important aspects of this module is to be able to distinguish between match and no-match pages resulting from the submission of the form. The distinction between match and no-match pages can be made using a number of heuristics (the size of the no-match page is smaller, there are less outgoing links, presence of keywords like *Error* or *No match*, absence of keywords such as *Next* or *More*). We choose to use a much more robust approach by performing a clustering of result pages obtained through probing. If a page is in a different cluster than the no-match page obtained with the submission of a nonsense word, the page is labeled as a match page.

```
Algorithm 2 Confirming field annotations with probing
Input: A given field of a form together with its probabilistic
annotations.
Output: A confirmed annotation for this field, or none.
(a) First, identify an error page by probing the field with a
    nonsense word.
(b) Let c be the concept with the highest probability the
    field is annotated with.
(c) Probe the field with a set of instance words, randomly
    chosen according to their frequency distribution, to get
    a corresponding set of pages.
(d) Cluster the set of pages obtained by probing with the
    error page, using an incremental clustering algorithm on
    terminal paths in the DOM tree of the pages.
(e) If some pages obtained by probing are different from
    the result page, confirm the annotation; otherwise, retry
    with the next best concept annotation.
```

We use an *incremental clustering* algorithm, which works well in our context since we have a small number of documents to cluster, with significant differences between error and result pages. The feature vector representing an HTML page for clustering is built as follows: In the DOM tree of an HTML document, *Terminal paths* are the set of paths from the root to a leaf of the tree. Each distinct sequence of node labels (that is, HTML element names) along a terminal path forms a dimension of the vector space that we use for clustering. Each page is then represented in this vector space with a *tf-idf* (term frequency / inverse document frequency) measure, depending on which terminal paths are present in the DOM tree. Finally, the cosine similarity measure is used to compare any two vectors during clustering. The idea is that two result pages share most of their terminal paths, some of them may just be repeated more often than others; on the other hand, a result page and an error page have significantly different structures (no list of results appears in an error page) that leads to a completely different representation in the vector space of terminal paths in the DOM tree.

Note that we may get more than two clusters using our clustering algorithm, if there are two or more significantly different kinds of result pages. This is the case, for instance, with the publication database DBLP, where searching for an ambiguous author name results in a different page than searching for a name that only appears once in the database. It is then important to use multiple words for probing, representative of their frequency distribution, so as to (i) generate all possible kinds of result pages; and (ii) be sure to get a result page, as long as the service probed has similar content to our domain knowledge.

4. EXTRACTING RESULT PAGE DATA

Each cluster identified in the probing step represents a specific class of result pages. All pages of a class have a similar structure and typically display repetitive sections, such as items of a list or rows in a table; each such item stands for one possible query result, or a *record* of the database. As our goal is to understand the result pages, and thus to be able to identify records and their corresponding domain instances, we need to somehow (i) find the repetitive parts on a page (e.g., the articles); and (ii) determine the internal structure of these records that defines the position of the domain instances (title, authors, publication date, etc.). The general process that we propose for this is shown in Figure 2 and detailed in this section. Several approaches exist to recognize the variable parts of a page (which may correspond to the records) in an unsupervised way (e.g., [6]), but most lack the ability to map the extracted fields to the domain concepts and thus in the end depend on some user input.

We approach the problem from the other side: instead of first regarding the structure, we rely on our domain knowledge to annotate HTML text nodes with domain concepts. This is done using a *gazetteer*, i.e., a dictionary of possible domain instances created with the domain knowledge (e.g., lists of author names, known article titles or journals). This results in a very raw, incomplete, and faulty, textual annotation.

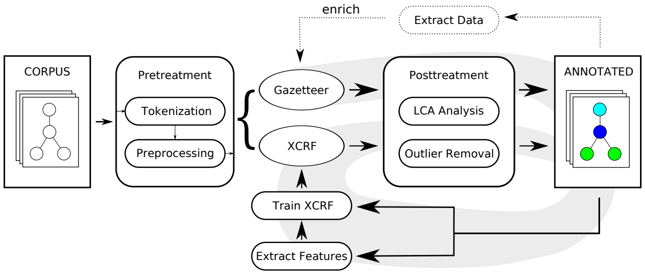

Figure 2: Result page extraction process

However, if we interpret the HTML document as its DOM tree, we can use a probabilistic discriminative model to infer structural relationships between the annotations, and thus use the repetitive structure to improve our initial annotation. We have elaborated a flexible model working on XML documents (XCRFS, [12]) which can easily be applied to well-formated XHTML documents. It is a special form of a Conditional Random Field [13] which models dependencies between parent, child and sibling nodes, conditioned by features possibly ranging over the entire input document. The probability of annotation $\mathbf{y}$ given input $\mathbf{x}$ is $p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_C \sum_k \lambda_k f_k(\mathbf{x}, \mathbf{y}_C, C)\right)$ where Z is a normalization factor, C are the 3-cliques (parent, child and sibling nodes) of the graph and f_k are feature functions evaluated on a clique C weighted by λ_k. They are of the form

$$f_k(\mathbf{x}, \mathbf{y}_C, C) = \begin{cases} 1 & \text{if} \quad y_{\text{parent}} = \mathit{label}_1,\ y_{\text{this}} = \mathit{label}_2, \\ & \qquad y_{\text{sibling}} = \mathit{label}_3,\ p(\mathbf{x}, C); \\ 0 & \text{otherwise.} \end{cases}$$

The predicate $p(\mathbf{x}, C)$ is an XPath expression evaluated in the context of the clique C. We can thus model all kinds of conditions (e.g., label some node as `Title` depending on the annotation of its father, only if it is a `<td>` node with attribute `class="author"`), provided that the information is accessible via XPath expressions. Luckily, almost all kinds of information can be introduced into the XHTML model by adding attributes to tree nodes in a preprocessing step.

As the nature of the input documents is forcedly unknown in an unsupervised context, these conditions, which refer directly to the tag names and attributes, cannot be predefined or, even worse, provided by the user. Instead, a few templates combined with a neighborhood definition are fixed (e.g., tag name or attribute values of grandparent, parent, siblings) and the explicit values and label conditions are filled in by observing the training data. For example, if a node annotated as `Author` is found, the tag names of its grandparent, parent and sibling are checked and the corresponding three f_k's are created. This defines a procedure to generate features in a completely unsupervised way. As this produces a great magnitude of possible features, a preliminary sorting out of features with little support is performed.

Once the model is defined by the automatic feature generation process, it must be trained to determine the feature weights λ_k. The input sample is obtained by the gazetteer annotation. Unfortunately, the annotation is imprecise: precision as well as recall of the gazetteer annotation are often below 0.5. To obtain meaningful results, we have to subject our naively annotated input documents to heuristic algorithms, filtering out very unlikely annotations before using them as training data for our probabilistic model. We have implemented a simple algorithm to identify records in the HTML tree using LCAs (least common ancestors) of annotated nodes and *suffix trees* (more details in [17]). This allows us to (i) eliminate all annotations not belonging to a record and (ii) find the annotations belonging together to define a record, which is necessary for the extraction step following the annotation. After this heuristic cleanup, we select those records which contain a significant number of annotated nodes and use them to train the XCRF. It is worth noting that the same heuristics used to prepare the training data can also be applied to the annotations resulting from an annotation by the XCRF to improve the overall quality and to allow for extraction. Results presented in Section 6 all include this cleanup.

The trained model can then be used to find an annotation $\hat{\mathbf{y}}$ of a new input page $\mathbf{x}$ with maximum likelihood $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. We obtained annotations of mainly higher quality than that of the gazetteer and can use them to extract the data from the pages. We can also use this information to enhance our domain knowledge, and thus the gazetteer. This leads to a bootstrapping process, where a better gazetteer leads to better trained XCRFs and thus to better information extraction and further enhancement. Furthermore, if this process is extended to include more than one result page class (e.g., not only the results of CiteSeer, but also of GoogleScholar and ACM), we can use the inherent redundancy of information to filter out unlikely data or enhance incomplete data. If our domain knowledge is not sufficient to allow for a good gazetteer annotation of one source, we can first enhance it with another source and then use the additional information to exploit the first source. However, this requires data integration techniques and heuristics to determine the quality of the information extracted from the sources, and thus be able to decide which data should be used for learning, and which should be rejected as uncertain.

5. WRAPPING FORMS AS WEB SERVICES

Once the input fields of a given query form are identified as well as the output understood, the human-centric HTML interface can be wrapped as an abstract Web-service, which communicates its input and output format in some machine-readable form. One way of describing such a service is using a WSDL [29] document, implemented as a Java servlet. All services wrapping individual query forms are then integrated in one main service offering relevant domain concepts as inputs and transmitting the user request to the selected service. A user (whether human or machine) can thus query different services such as GoogleScholar or DBLP with the same well-defined interface, and get back structured results. This is is done by integrating the two systems described in Sections 3 and 4 in the following way. Given the URL of a form, this form is analyzed and probed as described in Section 3. This also results in the decomposition into clusters of generated result pages. Inside each of these clusters, pages are annotated by the gazetteer and serve as the training set for a separate XCRF wrapper, as described in Section 4. Then, the user is presented with the concepts that were found in the form analysis phase. When submitting a query, the concepts are translated into the corresponding fields of the original form, the form is submitted, the cluster of the resulting page is identified, and the appropriate XCRF wrapper is used to extract the data and present it to the user.

6. EXPERIMENTS

The components of the system presented in this paper have been implemented in either Java or Perl, and we report here on the experiments carried out to validate the approach. A set of ten publication database services of the hidden Web were selected by hand (with the help of a classical search engine), the list of which is given in the first column of Table 1; these databases are either specific to computer science or generalist. Each corresponding URL was then fed to the system, which resulted, after processing by the different modules, into a wrapped service, as described in Section 5. Because of the limitations of our system for probing the hidden Web described in Section 3, a different approach was followed for two of the sources, namely ACM (the *advanced search* form uses complex combinations of fields for expressing a concept/value relation) and CiteSeer (there is no *advanced search* feature). For these two sources, a few result pages were generated by manual queries, and the gazetteer annotation and XCRF wrapper induction was performed on these pages, as described in Section 4. Otherwise, five probes per pair of field and candidate concept annotation were performed and, when found, “Next” links were followed to retrieve a maximum of five HTML result pages.

The testing process of the induced wrappers was then the following: For each source, a *perfect* wrapper of its result pages was manually written for comparison[1]. We then arbitrarily generated additional result pages and compared the result of both the gazetteer and the learned XCRF wrapper with the reference annotation given by the perfect wrapper.

Experimental results are presented in Table 1. Let us first focus on the columns under the heading **Query form**, that are an evaluation of the quality of the analysis of the HTML form by the heuristics and probing described in Section 3. For each source, the precision (ratio of fields correctly annotated over all annotated fields) and recall (ratio of fields correctly annotated over all relevant fields) of both the initial annotation (cf. Algorithm 1) and the step of confirmation by probing (cf. Algorithm 2) are given. They are computed with respect to a *perfect* annotation of the form, manually elaborated. The first observation is that, despite the various assumptions that we made on the fields of a form, we still get quite good results; in particular, the average precision for our dataset is 82 %, while the average recall is 73 %. Besides, the precision reaches 100 % in a majority of cases. The other observation that can be made is that the probing and confirmation step is indeed very useful, since it removes a large number of incorrect annotations. Indeed, this probing step has for effect, for all considered sources, to raise the precision of the annotation *while keeping the same recall*. This may obviously not always be so, the probing step may very well reduce the recall in some cases, especially if the database is small, but it is interesting to note that this did not happen in the experiments. It is a validation of the probing and clustering approach. Note also that the perfect annotation of DBLP is not really an artifact of our choice of DBLP as domain knowledge (since we only use concept names and no concept instances in our initial form annotation), but rather due to the good structure and simplicity of the DBLP search form.

The observed quality of form analysis is all the more interesting since the methods that we used are quite basic and subject to all kinds of refinements. Improving even more the precision should perhaps be easier than improving recall: An idea is to be more cautious and less tolerant during

[1] This task was actually quite time-consuming, even with the full expressive power of a programming language; besides, defining the perfect annotation of a given result page may be ambiguous.

Table 1: Experimental results of the analysis of the query form and result pages of some publication databases.

	Query form				**Response page**					
	Initial annot.		Confirmed annot.		`Title`		`Author`		`Date`	
	p (%)	r (%)	p (%)	r (%)	F_g (%)	F_x (%)	F_g (%)	F_x (%)	F_g (%)	F_x (%)
ACM					77	100	86	94	97	100
Cambridge	22	67	67	67	75	83	69	63	86	57
CiteSeer					54	79	59	68	78	68
Citebase	100	67	100	67	13	47	49	59	33	44
DBLP	100	100	100	100	87	96	84	95	96	95
DivaPortal	50	50	100	50	4	0	72	92	93	100
Elsevier	25	100	100	100	32	4	56	79	97	100
GoogleScholar	33	73	40	73	51	56	49	32	80	26
IngentaConnect	50	100	100	100	13	80	59	53	91	88
IowaState	11	25	50	25	34	86	54	64	100	83
Average	49	73	82	73	44	63	64	70	85	76

p: precision; r: recall; F_g: F_1-measure for the gazetteer annotation; F_x: F_1-measure for the XCRF annotation.

the probing step, only probing with words that are unambiguously attached to a given concept, while requiring that most probes return result pages. This might, however, reduce the coverage quite a lot. Improving the recall may be quite hard, in the situation where the textual context of a field is not descriptive enough to get an annotation. We may try, however, in these cases, to probe a field with each concept in turn; as the number of fields and concepts are small enough, this seems feasible. Note finally that the time required for all this processing is essentially the network access times required for the probes, all other operations taking a negligible time.

As explained in Section 3, the feature vector that we use for clustering is the set of terminal paths in the DOM tree of the document, with tf-idf weighting. The DOM tree captures the structure of the document perfectly, and works particularly well for our experiments. For instance, the cosine similarities between the result pages from GoogleScholar are up at around 0.99, whereas the similarities between result and error pages are of the order of 0.01. To show that the DOM tree model is an adequate choice, we also experimented with a feature vector based simply on the occurrence of HTML tags in the document [4]. We simply consider all tags that occur in the document, compute the tf-idf score based on the occurrence of tags in the collection and use the cosine similarity between these vectors for clustering. It was found that this approach assigns a rather high degree of similarity between result and error pages (for GoogleScholar, it was of the order of 0.5 to 0.6, for instance, which makes the clustering process very dependent of the threshold).

Consider now the columns under the heading **Response page** of Table 1. For each of the three most occurring concepts in the result pages of publication databases, namely `Title`, `Author`, and `Date`, the precision p and recall r of the annotations obtained both by the gazetteer and by the learned XCRF wrapper over a set of sample pages are summarized with the standard F_1-measure defined as $F_1 = \frac{2 \cdot p \cdot r}{p+r}$. Here, precision and recall are measured in terms of the numbers of tokens whose annotation is the same as with the perfect wrapper. When result pages of a single source belonged to multiple clusters, only the results on the most commonly occurring cluster are shown.

Note first that absolute values of the F_1-measures, between 60 % and 80 % are quite acceptable, in the absence of any other fully automatic alternative for assigning concepts to parts of the result pages. With the notable exception of the concept `Date`, the structural XCRF wrapper performs generally better than the gazetteer. This means that the XCRF wrapper was not only capable of reproducing the gazetteer annotation (this was not guaranteed, since the wrapper does not have access to the domain knowledge, but only to structural features), but also of improving it by filtering out outliers or adding missing annotations thanks to a structural generalization. This shows that it is indeed possible to use supervised machine learning techniques in an unsupervised way with the help of a noisy annotation, to induce a structural wrapper that performs better than this noisy annotation. There are exceptions to this, however, which are interesting in themselves, as detailed below.

In cases where the gazetteer performs badly (see for example, `Title` for DivaPortal and Elsevier), the XCRF wrapper performs even more badly, because it is not able to find any structural generalization of the annotation with enough support. Recall that the gazetteer only annotates titles when it finds the same exact title as an article appearing in DBLP records. This is not a very elaborate scheme, and it fails when the content of the database is significantly different from that of the domain knowledge : the pages from Elsevier from which the learning was made were semantically too far away from computer science, while most publications from DivaPortal have a Swedish title, which has no chance of appearing in the domain knowledge. However, even with such a naïve way of recognizing titles, the structural pattern learned by the wrapper is in all other cases better (and sometimes much better, see for instance IngentaConnect and IowaState) than the original gazetteer annotation. The case of the concept `Date` is special, as the gazetteer already performs very well due to the relative non-ambiguity of dates as lexical tokens, while their position in the document structure is often not easy to isolate. This means that, for such a concept, there is not much (if anything) to be gained by this structural learning, and that classical date entity recognizers are often enough. Some result pages do not make much use of the structuring features of HTML and basically present the different fields of each publication in a linear, textual, way. In such conditions, it can be very difficult to isolate each field in a structural manner; this was also reflected by the complexity of writing the corresponding perfect wrappers. In such cases (see, for instance, Author and Date for GoogleScholar or Cambridge), a simple gazetteer annotation, which already gives quite good results, is enough. Perhaps a direction for future work would be to identify in an automatic way such cases in advance, and not to try any structural generalization then. On the other hand, on highly structured result pages such as those from ACM, the generated XCRF is close to perfection.

We would like to stress that one of the major advantages of the learned XCRF wrapper over the gazetteer is that it is mostly independent of the considered subdomain; this means that if such a wrapper has been learned for a generalist database such as GoogleScholar on a set of computer-science–related result pages generated by the prober, it can then be used on any result page of this source (as long as it has the same structure and can be handled by the same structural wrapper), while the gazetteer is utterly unable to annotate titles of, say, theoretical physics articles.

7. RELATED WORK

An early work on crawling the hidden Web is [22], where Raghavan and Garcia-Molina present a system that focuses on analyzing Web forms, automatically generating queries and extracting information from the response pages thus obtained. There are many similarities between the approach described in [22] and our initial structural analysis of an HTML form, though the authors choose to use a method based on the visual layout of elements to determine the label of a field, rather than the more structural method that we use. The MetaQuerier system similarly processes forms with multiple attributes [33] and uses predicate mapping to convert user-given queries to specific form queries, and thus fetches the required response pages hidden behind the form interface. The focus of this work is on schema mapping and query rewriting to translate queries to a given interface. The paper does not address the analysis of forms themselves, but the same authors describe in [32] a fairly elaborate approach, based on the notion of *hidden grammars* that describe the relation between structure of a query form and its visual layout. [11] focuses on the sampling of a source of the hidden Web, by using domain knowledge to obtain a representative subset of result documents. [1] is another example of a system which extracts hidden-Web data from keyword based interfaces by querying the interface with high coverage keywords. The aim of the authors is to extract all the data from the database, and index it locally, which is quite a different goal. They exploit the same idea as we do for distinguishing between result and error pages, citing [7] as their inspiration, that is, probing the form with *nonsense* keywords that we are sure do not exist in the database. Finally, the idea of using clustering for distinguishing between result and error pages comes from [4], although we do not use the same input for the clustering algorithm. In [4], the authors construct the feature vector for a page by extracting the tags from HTML code and use the cosine similarity measure with a tf-idf weighting. In practice, we found out that this tag-signature–based clustering does not work very well in comparison to our scheme of clustering based on the terminal paths in the DOM tree.

Early works on machine learning approaches to Web information extraction deal with the supervised approach. It is supposed that domain instances of a target domain concept are first manually annotated by an expert. Then machine learning techniques are applied to generate an extraction program. Several systems explicitly consider the tree structure of Web documents with different machine learning techniques, e.g., Stalker [18], Squirrel [3], Lipx [27]. Later on, the unsupervised approach was introduced, systematically trying to avoid manual labeling [31]. However, they are less accurate than supervised systems. Moreover a manual post-processing is needed to map the extracted fields to the domain concepts because of the inability of the extraction programs to understand the semantic of extracted data. In the Natural Language Processing (NLP) community, discriminative probabilistic models have been successfully applied to a number of information extraction tasks in supervised systems [25, 15, 19, 23]. Most approaches use models for sequences whereas we use models for trees in order to take profit of the tree structure of result pages. For instance, a generative model of sequences is used in [9] whereas a discriminative model is used in [14]. In the latter, a small set of labeled sentences is required whereas in the former the domain is restricted and labeled sequences are constructed by using many heuristics. It was recently proposed to use gazetteers in a discriminative model [26] containing user-defined features. Gazetteers are used to define additional features. We follow this approach but we use gazetteers to define an initial approximate annotation which replaces the manual annotation, i.e., we use gazetteers to use discriminative models in an unsupervised context. Moreover, we use a discriminative model for tree-structured documents rather than for text. Two aspects of our work on information extraction have some similarities with papers about the MetaQuerier system. In [30], bootstrapping a knowledge base by extracting information on result pages and injecting them back on subsequent probing is discussed. Bootstrapping across different sources, in order to benefit from their different coverage, is the topic of [5]. Our main idea of using supervised techniques on the structure of a document to generalize an annotation by a gazetteer has not been explored in either of these works.

8. CONCLUSIONS

We have presented an original, standalone, approach to automatic wrapper induction from sources of the hidden Web, with the help of domain knowledge. It does not need any human supervision and relies on a combination of heuristics, clustering, gazetteer annotation and machine learning. Experiments show that this approach is able to understand quite well the structure of a form, and that there is potential in the use of machine learning to obtain a structural generalization and correction of an imperfect, imprecise, annotation. It is our belief that, as illustrated here in the case of hidden-Web services, exploiting the structure of content to help correct or disambiguate a purely linear text-based analysis is a fruitful idea. Another important pattern is shown in the two-step process of our form analysis and probing module, the first step with high recall but possibly low precision, and the second step raising precision without hurting recall.

There are a number of directions that can be followed in relation to this work. First, this system is to be thought as part of a general framework for the understanding of the hidden Web, that would include service discovery, semantic analysis of the relations between input and output concepts of a service, and indexing and high-level querying of semantically analyzed services. We are already working on some of these problems. Second, a number of improvements could be made to our system: a number of the assumptions made on the structure of the forms could be removed by more elaborate structural analysis, or the gazetteer could be improved to recognize titles as linguistic entities of a certain form, rather than as fixed strings. Finally, a perhaps deeper issue would be to improve the learning step itself, that is at the moment partially hindered by the fact that conditional random fields, as all supervised machine learnings techniques that we know of, assume that the original annotation is perfect and try to fit it as much as possible. An adapted machine learning model would consider the description length of the corresponding wrapper as something as important to minimize as the fitting to the annotated dataset.

9. ACKNOWLEDGMENTS

We would like to acknowledge Serge Abiteboul, Patrick Marty, Fabien Torre, and especially Florent Jousse, for their comments and participation in earlier versions of this work. This work was partially supported by the projects Atash ANR-05-RNTL00102 and Webcontent ANR-05-RNTL02014.

10. REFERENCES

[1] L. Barbosa and J. Freire. Siphoning hidden-Web data through keyword-based interfaces. In *Proc. Simpósio Brasileiro de Bancos de Dados*, Brasília, Brasil, Oct. 2004.

[2] BrightPlanet. The deep Web: Surfacing hidden value. White Paper, July 2001.

[3] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducers. *Machine Learning Journal*, 66(1):33–67, Jan. 2007.

[4] J. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep Web. In *Proc. ICDE*, Boston, USA, Mar. 2004.

[5] S.-L. Chuang, K. C.-C. Chang, and C. Zhai. Context-aware wrapping: Synchronized data extraction. In *Proc. VLDB*, Vienna, Austria, Sept. 2007.

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

[7] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *Proc. Agents*, Marina del Ray, USA, Feb. 1997.

[8] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proc. AAAI*, Austin, USA, July 2000.

[9] J. Geng and J. Yang. AUTOBIB: Automatic extraction of bibliographic information on the web. In *Proc. IDEAS*, 2004.

[10] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep Web: A survey. *Communications of the ACM*, 50(2):94–101, May 2007.

[11] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden Web: Hierarchical database sampling and selection. In *Proc. VLDB*, Hong Kong, China, Aug. 2002.

[12] F. Jousse, R. Gilleron, I. Tellier, and M. Tommasi. Conditional Random Fields for XML trees. In *Proc. ECML Workshop on Mining and Learning in Graphs*, Berlin, Germany, Sept. 2006.

[13] J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML*, Williamstown, USA, June 2001.

[14] I. Mansuri and S. Sarawagi. A system for integrating unstructured data into relational databases. In *Proc. ICDE*, 2006.

[15] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields. In *Proc. CoNLL*, Edmonton, Canada, May 2003.

[16] A. Mittal. Probing the hidden Web. Research internship report. Technical Report 479, Gemo, INRIA Futurs, July 2007.

[17] D. Muschick. Unsupervised learning of XML tree annotations. Master's thesis, Université de Technologie de Lille and Technischen Universität Graz, June 2007.

[18] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Proc. AAMAS*, 4(1–2):93–114, 2001.

[19] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proc. SIGIR*, Toronto, Canada, July 2003.

[20] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[21] Princeton University Cognitive Science Laboratory. WordNet. `http://wordnet.princeton.edu/`.

[22] S. Raghavan and H. Garcia-Molina. Crawling the hidden Web. In *Proc. VLDB*, Roma, Italy, Sept. 2001.

[23] S. Sarawagi and W. W. Cohen. Semi-Markov conditional random fields for information extraction. In *Proc. NIPS*, Vancouver, Canada, Dec. 2004.

[24] P. Senellart. *Comprendre le Web caché. Understanding the Hidden Web*. PhD thesis, Université Paris-Sud 11, Orsay, France, Dec. 2007.

[25] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. HLT-NAACL*, Edmonton, Canada, May 2003.

[26] A. Smith and M. Osborne. Using gazetteers in discriminative information extraction. In *Proc. CoNLL*, New York, USA, June 2006.

[27] B. Thomas. Bottom-up learning of logic programs for information extraction from hypertext documents. In *Proc. PKDD*, Catvat-Dubrovnik, Croatia, Sept. 2003.

[28] W3C. HTML 4.01 specification, Sept. 1999. `http://www.w3.org/TR/REC-html40/`.

[29] W3C. Web Services Description Language (WSDL) 1.1, Mar. 2001. `http://www.w3.org/TR/wsdl`.

[30] W. Wu, A. Doan, C. T. Yu, and W. Meng. Bootstrapping domain ontology for semantic Web services from source Web sites. In *Proc. Technologies for E-Services*, Trondheim, Norway, Sept. 2005.

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

[32] Z. Zhang, B. He, and K. C.-C. Chang. Understanding Web query interfaces: best-effort parsing with hidden syntax. In *Proc. SIGMOD*, Paris, France, June 2004.

[33] Z. Zhang, B. He, and K. C.-C. Chang. Light-weight domain-based form assistant: Querying Web databases on the fly. In *Proc. VLDB*, Trondheim, Norway, Sept. 2005.