Information Extraction from Tree Documents by Learning Subtree Delimiters

Boris Chidlovskii
Xerox Research Centre Europe, France
6, chemin de Maupertuis, F–38240 Meylan, chidlovskii@xrce.xerox.com

Abstract

Information extraction from HTML pages has been conventionally treated as plain text documents extended with HTML tags. However, the growing maturity and correct usage of HTML/XHTML formats open an opportunity to treat Web pages as trees, to mine the rich structural context in the trees and to learn accurate extraction rules. In this paper, we generalize the notion of *delimiter* developed for the string information extraction to tree documents. Similar to delimiters in strings, we define delimiters in tree documents as subtrees surrounding the text leaves. We formalize the wrapper induction for tree documents as learning the classification rules based on the subtree delimiters. We analyze a restricted case of subtree delimiters in the form of simple paths. We design an efficient data structure for storing candidate delimiters and an incremental algorithm for finding most discriminative subtree delimiters for the wrapper.

1 Introduction

The immensity of Web data valuable for various human needs has led to research on information extraction from the Web, with the wrapper learning from annotated samples being one of major research trends. Since the first wrappers [10] crafted for a specific structure of Web pages, wrapper classes have grown in their expressive power and capacity to adopt structural variations. While the further empowering the wrapper learning methods and their combinations remains crucial for developing flexible IE systems, another important goal raises in the controlled reduction of the sample annotation. The learning from both labeled and unlabeled samples appears, in the case of the wrapper learning, as the learning from partially annotated Web pages, where the annotation of items in a page is integrated with the learning in an interaction system and driven by the learning bias and accuracy requirements.

Over last 10 years, the HTML format has seen several evolutionary changes and has achieved a maturity level with a wider use of XHTML/XML for publishing the Web content. In November 2002, we have analyzed HTML pages from 32 sites we have being tracked since 1998 (360 to 420 pages per year). The analysis has discovered a tendency toward cleaner pages and richer tag context around content elements. First, the nesting error ratio expressed as the percentage of missing and mismatching end tags in the HTML files has almost halved, from 6.7% in 1998 to 3.9% in 2002. Second, the average number of HTML tags surrounding a content element has increased by 31%, from 5.1 tags per content element in 1998, to 6.7 tags in 2002. Additionally, the ratio of tag attributes has increased by 26%, from 0.34 attribute per tag in 1998 to 0.43 in 2002.

Although it seems very natural considering Web pages as trees, the majority of the wrapper learning methods treat HTML pages as sequences of tokens where text tokens are interleaved with tags. Information extraction from strings often follows the finite-state methodology with two alternative approaches seen as the global and the local view at the extraction problem. The *local view* approach stems from the information extraction from unstructured and semi-structured text [5], when a wrapper is an enhancement of a basic HTML parser with a set of extraction rules; an extraction rule has often a form of delimiters (landmarks) [9; 11] that are sequences of tags preceding (or following) an element to be extracted; for example, delimiter `<td><a>` requires a text token to be preceded by tags `<td>` and `<a>`.

The *global view* approach assumes that HTML pages are instances of an unknown language and attempts to identify this language. In the case of deterministic automata, it determine the automata structure by generalization from the training examples; in the case of weighted automata/HMM, it learns the transition probabilities. To accommodate the information extraction, these methods either enhance finite-state automata with extraction rules [4] or adopt the formalism of finite-state transducers [2; 7].

The global view approach benefits from the grammatical inference methods that can learn finite-state automata and transducers from positive examples; however they often require many annotated samples to achieve a reasonable generalization. On the other hand, in the local view, using local delimiters in a context-less manner limits the expressive power of the delimiter-based wrappers. To combine the advantages of the two approaches, [2] has extended the notion of delimiter to previously labeled text tokens. For example, delimiter `PC(none)<td><a>` requires that a current text token is preceded by a text token labeled as `none` (skipped) and tags `<a>` and `<td>`. As result, the wrapper learning algorithm

produces a set of extraction rules equivalent to a minimal regular transducer that can be obtained following the global view approach.

Information extraction from HTML trees consists of classifying the tree leaves with classification labels from a set C. The shift from strings to trees considerably enlarges the number of tags "surrounding" a text token and enriches the notion of tag proximity w.r.t. a similar proximity in strings which are a specific traversal of HTML trees.

So far, little research addresses the information extraction from tree documents [3; 6; 8]. Some researchers study languages for wrapping tree structures and their expressive power [6]; other researchers develop learning algorithms for extraction from tree structures [3; 8]. Interestingly, wrapper "builders" in [3] fit the local view approach, while tree automata in [8] follow the global view approach. In the grammatical inference, certain results has been successfully extended from strings to trees [15], allowing to learn tree automata and context-free grammars from examples; however, more research is needed to achieve the same level of robustness and practical utility.

On the other hand, recent research in graph search and the Web mining [13; 14] offers novel methods for the mining of tree documents, in particular, finding the most frequent subtrees and tree patterns.

2 Information extraction from trees

We represent HTML/XML documents as unranked ordered trees. For badly formatted HTML documents, structure checker programs like HTML Tidy free utility from W3C[1] can detect missing and mismatching end tags and map an HTML source into a complete tree.

In an HTML/XML tree, inner nodes determine the structure of the document, and the leaf nodes and the tag attributes provide the document content. We follow [12] in abstracting HTML/XML documents as the class of unranked labeled rooted trees. Tree t is defined over an alphabet Σ of tag names. The set of trees, denoted by T_Σ, is inductively defined as follows:

1. every $\sigma \in \Sigma$ is a tree (leaf),
2. if $\sigma \in \Sigma$ and $t_1, t_2, \ldots, t_n \in T_\Sigma$, $n \geq 1$, then $\sigma(t_1, t_2, \ldots, t_n)$ is a tree in T_Σ.

There is no a priori bound on the number of children of a node in a tree; such tree are therefore unranked. For the sake of convenience, we present our method for the binary trees. Unranked trees can be encoded into binary tree in several ways and we adopt an encoding from [12] as one preserving the adjacency relationship between children of an inner node. Intuitively, the first child of a node remains the first child of that node; the other children become right descendants of the first child in the encoding. Whenever there is a right child but no left child, a leaf # is inserted. Also, when there is only a left child, a leaf # is inserted for the right child.

Example 1 *Consider the Database and Logic Programming site* [2] *(DBLP) and information extraction from its answers to title-relevant queries. An HTML fragment of an answer is shown in Figure 1. Each answer item on the page contains a title, one or more authors, conference and pages. The unranked tree for the HTML source fragment with the text values associated with the tree leaves is in Figure 2.a*[3]*. Figure 2.b shows the binary tree encoding of the unranked tree, as well as the annotation of leaves with classification labels. Binary trees like one in Figure 2.b serve as the training set for the wrapper learning system.*

Search Results for 'tree automata'

...

- Leon S. Levy, Aravind K. Joshi: Some Results in Tree Automata. STOC 1971: 78-85

...

Figure 1: HTML fragment from DBLP.

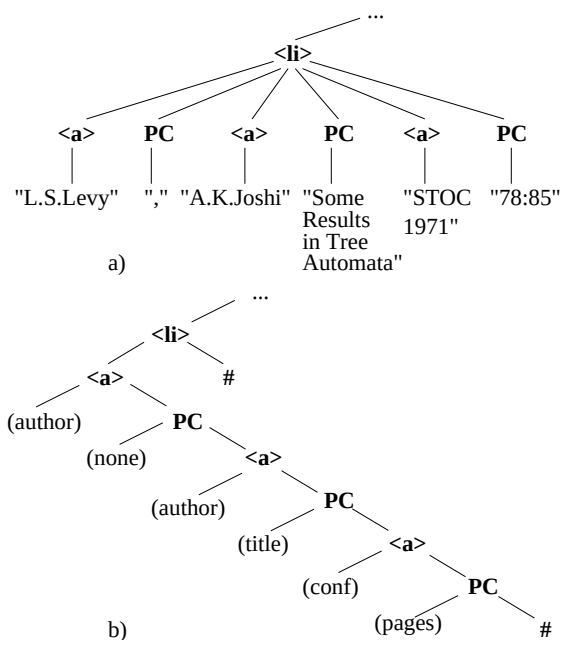

Figure 2: a) Unranked tree; b) its binary tree encoding and annotation.

Set of encoded ranked trees is denoted $T^2_{\Sigma'}$, where $\Sigma' = \Sigma \cup \{\#\}$.To introduce the notion of subtree, we allow a wildcard '*' to match any tree $t \in T^2_{\Sigma'}$. Tree s is a *subtree* of binary tree $t \in T^2_{\Sigma'}$, denoted as $s \lhd t$, if there exists a node n in tree t such that one of the following (inductive) conditions is satisfied:

- s='*',
- $s = \sigma$, $n = \sigma'$ and $\sigma = \sigma'$,
- $s = \sigma$, $n = \sigma'(t_1, t_2)$, and $\sigma = \sigma'$,

[1] Available at http://www.w3.org/People/Raggett/tidy/.

[2] www.informatik.uni-trier.de/∼ley/db/index.html.

[3] Special symbol PC is used for text fragments in the mixed elements.

- $s = \sigma(t_1, t_2)$, $n = \sigma'(t'_1, t'_2)$, and $\sigma = \sigma^{'}$ and $t_1 \lhd t'_1, t_2 \lhd t'_2$.

A *tree leaf delimiter* is a subtree associated with (at least) one annotated leaf. For the binary tree in Figure 2.b, binary subtree `PC (<a>(*,PC(*,#))` is a tree leaf delimiter associated with the `conf` leaf.

The information extraction from tree $t \in T^2_{\Sigma'}$ consists of labeling the leaves of t with classes from the set C, including `none` class for non-extracted leaves. A *leaf extraction rule* is a triple (s, c, cf) where s is a leaf delimiter, c is a classification label and cf is the confidence level, $0 < cf \leq 1$. A *tree wrapper* is a set of leaf extraction rules that are learned from a set of annotated documents. The tree wrapper works in the same way as the string wrappers do. In an unlabeled tree t, the tree leaves are visited in some traversal order and are tested with the delimiters of the extraction rules. If one or more leaf delimiters match the leaf context, the rule with the higher confidence is applied to label the leaf content. In Figure 2.b, leaf delimiter `PC(*,#)` discriminates the leaf `pages` with the confidence cf=1.0. Instead, the delimiter `<a>(*,PC)` is not discriminatory for the `author` leaf, as the same subtree `<a>(*,PC)` surrounds the `conf` leaf.

2.1 Simple path delimiters

The learning of tree wrappers requires exploring a large space of leaf delimiters and discovering such ones that provide the most accurate classification. In order to control the exponentially growing number of candidate delimiters in the trees, below we study and test an important subclass of tree wrappers, where leaf delimiters are *simple paths* in a binary tree.

A simple path delimiter is (topologically) equivalent to a simple path in a tree, i.e., all its nodes except the root have one child, and only the root may have one or two children. A simple path has two extremes, one extreme is the associated leaf and another extreme is an inner node or another leaf. The length of a simple path is given by the number of nodes in the path. One specific well-known case is *root-to-leaf paths* in a tree, when each leaf in the tree is associated with the path starting in the tree root and terminating in the leaf.

2.2 Candidate path index

To incrementally determine a good set of delimiters from annotated samples, we store the path candidates in a special data structure called *candidate index*. The goal of the index generation is two-fold. On one side, the index stores all delimiter candidates, as well as an additional information sufficient to determine the most discriminative delimiters. On the other side, the index allows to incrementally accommodate new annotated samples.

The candidate path index stores *reverse paths*, which are special encodings of simple paths; a reverse path starts at the annotated leaf and traverses tree arcs of three types; they are denoted Up ($\uparrow$), $Left$ ($\leftarrow$) and $Right$ ($\rightarrow$). For example, reverse path $\uparrow$ `PC` $\rightarrow$ # encodes the delimiter `PC(*,#)` of the `pages` leaf in Figure 2.b.

The index is a *trie data structure* where all nodes except the root are labeled with tags from Σ' and each transition is typed with $\uparrow$, $\leftarrow$ or $\rightarrow$. A path in the index from the root to any node corresponds to a reverse path and is a delimiter candidate. Additionally, each index node includes occurrences for all classification labels surrounded by the corresponding subtree in annotated documents.

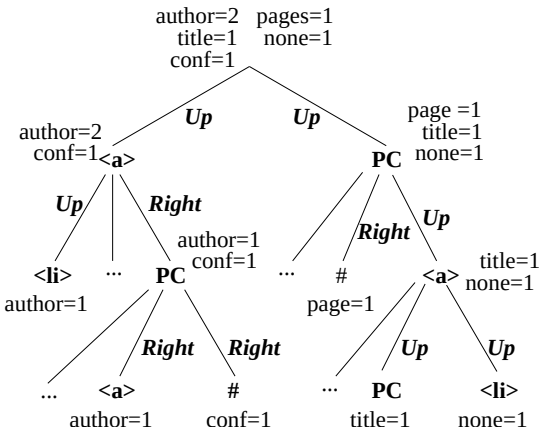

Figure 3: Fragment of path index for the example tree.

Figure 3 shows a fragment of candidate path index of the sample HTML fragment in Figure 2.b. In the index, node for the reverse path $\uparrow$ `<a>` indicates that the corresponding subtree surrounds two `author` leaves and one `conf` leaf. For the candidate index of the depth d=3, there exists a set of perfect extraction rules as follows:

($\uparrow$ `<a>` $\uparrow$ `<li>`, `author`,1),
($\uparrow$ `<a>` $\rightarrow$ `PC` $\rightarrow$ `<a>`,`author`,1),
($\uparrow$ `<a>` $\rightarrow$ `PC` $\rightarrow$ #, `conf`, 1),
($\uparrow$ `PC` $\leftarrow$ #, `pages`, 1),
($\uparrow$ `PC` $\uparrow$ `<a>` $\uparrow$ `<li>`, `none`, 1),
($\uparrow$ `PC` $\uparrow$ `<a>` $\uparrow$ `PC`, `title`, 1),

Like delimiters in the string wrappers, path delimiters are one-dimensional objects. However, unlike string wrappers, the algorithm exploits a larger (two-dimensional) space of simple paths in the tree in order to detect paths that are highly discriminative for the leaf classification but are "invisible" in the string presentation of HTML documents.

To manage the candidate index, we are currently using the simplest criteria, when the index depth is limited by some value $d > 1$. Figure 4 presents two routines for the index. Procedure $AddAnnotation()$ updates the index with the path candidates surrounding a new annotated leaf. Function $Classify(l)$ finds a classification label for a tree leaf. It implements the wide-first traversal through the intersection of the candidates in the index and simple paths starting in a leaf to be labeled.

3 Preliminary Experiments

We have tested the method of information extraction from tree documents using the experimental testbed developed for wrappers created with the XRCE Iwrap toolkit [1]. First, we have tested the tree wrappers on 14 "easy" sites (including

Let I be the candidate path index
define *AddAnnotation* (leaf l, class c):
$P(l, d)$:= set of reverse paths from l of max length d
for each path $p \in P(l, d)$ **do**
    if $p \in I$ **then**
        occurrence(p,c) := occurrence(p,c) + 1
    else
        n := the minimal prefix of p not in I
        add node n in I
        occurrence(n,c) := 1
endfor

define *Classify* (leaf l):
$candidateList$:={}
for i in 1, ..., d **do**
    $P(l, i)$:= set of reverse paths from l of depth i
    for each path p in $P(l, i)$ **do**
        if $p \in I$ **then**
            s, c, cf := rule in p with the highest confidence
            if cf = 1 **then return** c
            else add (c, cf) to $candidateList$
    endfor
endfor
return class c with the highest cf in $candidateList$

Figure 4: Two main routines for the candidate index.

Google, Altavista, Excite, CNN, ACM, Elsevier, DBLP Author, DBLP Title and some others) for which string wrappers in the form of regular transducers have been successfully learned (that is, with the F-measure superior to 98%); the wrappers manage to find out highly discriminative delimiters for all classification labels [2]. Second, we have tested the method on 6 "complex" sites including IEEE, CSbiblio, Medline and Cora from the Iwrap collection, and IAF and Shakespeare from the Kushmerick collections, for which the string wrappers obtain the average precision of 89.6% and recall of 84.3%.

For each site, 10 annotated documents were available for a series of 5 experiments. In each experiment, both string and tree wrappers have been trained from 3 randomly selected annotated pages and tested on other 7 pages. For each group of sites, we measure the precision/recall and the time a wrapper takes to parse a page and to classify the tree leaves. For the moment, we have run experiments with the maximal depth d=8 of the candidate index. The results of experiments are summarized in Table 1; the TRatio measure is a ratio of time a tree wrapper processes a page to the time the corresponding string wrapper processes the same page.

For 140 annotated HTML pages in the group of easy sites, an average page has 503.2 leaves to classify. Tree wrappers for these sites are as accurate as the string wrappers (see Table 1). However they are slower; on average, a tree wrapper spends 6.51 seconds to parse a page while a string wrapper spends 3.16 seconds only [4]. This happens because a tree wrapper verifies multiple paths surrounding a leaf against delimiters of the extraction rules.

[4]Experiments have been run on Sun Ultra 30 station under Solaris OS 8.

Sites	String wrappers		Trees wrappers		
	Prec	Rec	Prec	Rec	TRatio
Easy	100	98.8	99.8	99.1	2.06
Complex	89.6	84.3	96.3	92.6	3.21

Table 1: Comparison of string and tree wrappers.

For the group of complex sites, an average annotated page has 478.9 leaves to classify. Tree wrappers are here more efficient in finding accurate delimiters and obtain precision of 96.3% and recall of 92.6%. Instead, on average, a tree wrapper spends 3.21 more time to process a page than a string parser.

4 Conclusion

We have addressed the problem of information extraction from tree documents where inner nodes determine the document structure and the leaf nodes provide the document content. We have presented a method that learns the classification rules for the tree leaves in the form of subtrees. We have studied a special subclass of classification subtrees that have a form of simple trees and have adopted a trie data structure to ease the incremental management of annotated samples.

Preliminary experiments show that the tree wrappers are more efficient in finding discriminative tree delimiters that string wrappers, though this high accuracy is achieved by the cost of longer page processing. Further research may follow several directions. First, we want to investigate if finding more complex subtrees is beneficial for the leaf classification and, importantly, is compatible with the incremental management of annotated samples. Second, the current method does not assume any specific traversal of the document tree; introducing a traversal order and extending subtrees to previously annotated leaves may be as beneficial as with the string wrappers. Finally, a more efficient structure for managing annotated samples and delimiter candidates may reduce the page processing cost.

References

[1] Denis Bredelet and Bruno Roustant. Java IWrap: Wrapper Induction by Grammar Learning. Master's thesis, ENSIMAG, Grenoble, France, 2000.

[2] Boris Chidlovskii. Wrapping Web Information Providers by Transducer Induction. In *Proc. Europ. Conf. Machine Learning, Germany, Freiburg*, volume 2167 of *Lect. Notes Comp.Sci.*, pages 61–72. Springer, 2001.

[3] William Cohen and Lee Jensen. A structured wrapper induction system for extracting information from semi-structured documents. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, 2001.

[4] H. Davulcu, G. Yang, M. Kifer, and I.V. Ramakrishnan. Computational aspects of resilient data extraction from semistructured sources. In *ACM Symp. on Principles of Database Systems*, pages 136–144, 2000.

[5] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *Proc. AAAI/IAAI*, pages 517–523, 1998.

[6] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. In *Proc. ACM PODS*, pages 17–28, 2002.

[7] C.-N. Hsu and M.-T. Dung. Generating Finite-State Transducers for Semistructured Data Extraction from the Web. *Information Systems*, 23(8), 1998.

[8] Raymond Kosala, Jan Van den Bussche, Maurice Bruynooghe, and Hendrik Blockeel. Information extraction in structured documents using tree automata induction. In *Principles of Data Mining and Knowledge Discovery, 6th European Conference, Helsinki, Finland, LNAI 2431*, pages 299–310, 2002.

[9] N. Kushmerick. Wrapper Induction: Efficiency and Expresiveness. *Artificial Intelligence*, 118:15–68, 2000.

[10] N. Kushmerick, D.S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[11] I. Muslea, S. Minton, and C. Knoblock. A Hierarchical Approach to Wrapper Induction. In *Proc. the Third Intern. Conf. on Autonomous Agents Conference, Seattle, WA*, pages 190–197, 1999.

[12] Frank Neven. Automata Theory for XML Researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[13] D. Shasha, J. Tsong-Li Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *ACM Symp. on Principles of Database Systems*, pages 39–52, 2002.

[14] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):12–23, 2000.

[15] Y. Sakakibara. Recent Advances of Grammatical Inference. *Theoretical Computer Science*, 185(1):15–45, October 1997.