

A New Path Generalization Algorithm for HTML Wrapper Induction

Costin Bădică¹, Amelia Bădică², and Elvira Popescu³

¹ University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, RO-200440, Romania
`badica.costin@software.ucv.ro`

² University of Craiova, Business Information Systems Department
A.I.Cuza 13, Craiova, RO-200585, Romania
`ameliabd@yahoo.com`

³ University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, RO-200440, Romania
`elvira_popescu@software.ucv.ro`

Abstract. Recently it was shown that Inductive Logic Programming can be successfully applied to data extraction from HTML. However, the approach suffers from two problems: high computational complexity with respect to the number of nodes of the target document and to the arity of the extracted tuples. In this note we address the first problem by proposing an efficient path generalization algorithm for learning rules to extract single information items. The presentation is supplemented with a description of a sample experiment.

1 Introduction

The Web was originally designed as a major information provider for the human consumer, but the interest has rapidly shifted to make that information available for machine consumption. For example, Web directories and search engines are Web applications that are capable of providing useful information upon request to individual users, businesses or software agents.

However, despite the fact that technologies have been put forward to enable automated processing of information published on the Web (like semantic markup or Web services), most of the current practices in Web publishing are still being based on the combination of traditional HTML – *lingua franca* for Web publishing, with server-side dynamic content generation from databases. Moreover, many Web pages are using HTML elements that were originally intended for use to structure content (e.g. those elements related to tables), for layout and presentation effects, even if this practice is not encouraged in theory. Therefore, automatic information extraction from documents published on the Web has attracted a lot of researches during the last decade and this interest is expected to grow, as the Web is also growing in both size and complexity.

Information extraction (IE hereafter) is concerned with locating and extracting specific values in documents, and then using them to populate a database

or structured document. The IE research community has proposed a quite large variety of machine learning techniques for automatic information extraction ([7]).

Inductive Logic Programming – ILP is one of the success stories in the application area of wrapper induction for information extraction ([1, 2, 4, 7]). However, this approach suffers from two problems: high computational complexity with respect to the number of nodes of the target document and to the arity of the extracted tuples. In this paper we address the first problem by proposing a path generalization algorithm for learning rules to extract single information items (a task similar to [1]). The algorithm produces an XPath ([10]) extraction path from positive examples and is proven to have good computational properties. The presentation is supplemented with a detailed description of a sample experiment that shows how the technique performs in practice on real Web pages.

We proceed as follows. In section 2 we define extraction paths. In section 3 we describe an algorithm for learning extraction paths. In section 4 we show how extraction paths can be translated to XPath. In section 5 we describe an experiment showing our technique at work on a real Web site. In section 6 we present researches connected to our work.

2 Extraction Path

We model well-formed HTML documents as labeled ordered trees. An extraction path takes a labeled ordered tree and returns a subset of extracted nodes. An extracted node can be viewed as a subtree rooted at that node. The node labels of a labeled ordered tree correspond to tags in HTML texts. Let Σ be the set of all node labels of a labeled ordered tree.

For our purposes, it is convenient to abstract labeled ordered trees as sets of nodes on which certain relations and functions are defined. Figure 1 shows a labeled ordered tree with 25 nodes and tags in the set $\Sigma = \{a, b, c\}$.

An *extraction path* is a labeled directed graph. Arc labels denote conditions that specify the tree delimiters of the extracted information, according to the parent-child and next-sibling relationships (eg. is there a parent node ?, is there a left sibling ?, a.o). Vertex labels specify conditions on nodes (eg. is the tag label *td* ?, is it the first child ?, a.o). A special vertex of this graph is used for selecting the nodes for extraction.

Intuitively, an arc labeled '*n*' denotes the "next-sibling" relation while an arc labeled '*c*' denotes the "parent-child" relation. As concerning vertex labels, label '*f*' denotes "first child" condition, label '*l*' denotes "last child" condition and label $\sigma \in \Sigma$ denotes "equality with tag σ " condition.

Note that we use the term 'node' when referring to document trees and the term 'vertex' when referring to the graph of an extraction path.

Definition 1. (*Extraction path*) An extraction path is a labeled directed graph that can be described as a list $[t_0, t_1, \dots, t_k]$, $k \geq 0$ with the following properties:

1. Each element t_i , $0 \leq i \leq k$ is a list $[v_{-l}, \dots, v_{-1}, v_0, v_1, \dots, v_r]$, $l \geq 0$, $r \geq 0$ such that: i) v_i , $-l \leq i \leq r$ are vertices; ii) (v_i, v_{i+1}) , $-l \leq i < r$ are arcs

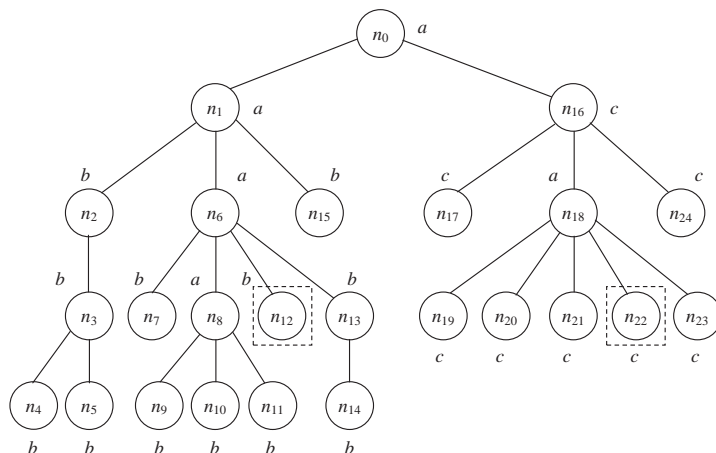


Fig. 1. Document as labeled ordered tree

- labeled with ' n' ', and iii) for each pair of adjacent lists t_i, t_{i+1} , $1 \leq i < k$ in the extraction path, (v_0^{i+1}, v_0^i) is an arc labeled with ' c '.
2. Vertex labels are defined as: i) if $l > 0$ then v_{-l} is labeled with a subset of $\{f', \sigma\}$, $\sigma \in \Sigma$; ii) v_i , $-l < i < 0$ is labeled with a subset of $\{\sigma\}$, $\sigma \in \Sigma$; iii) if $r > 0$ then v_r is labeled with a subset of $\{l', \sigma\}$, $\sigma \in \Sigma$; iv) v_i , $1 < i < r$ is labeled with a subset of $\{\sigma\}$, $\sigma \in \Sigma$; v) If $l, r > 0$ then v_0 is labeled with a subset of $\{\sigma\}$, $\sigma \in \Sigma$; if $l = 0, r > 0$ then v_0 is labeled with a subset of $\{f', \sigma\}$, $\sigma \in \Sigma$; if $l > 0, r = 0$ then v_0 is labeled with a subset of $\{l', \sigma\}$, $\sigma \in \Sigma$; if $l = r = 0$ then v_0 is labeled with a subset of $\{f', l', \sigma\}$, $\sigma \in \Sigma$.
 3. Vertex v_0^k (i.e. v_0 in list t_k) is matched against the extraction node and consequently is called extraction vertex.

It is not hard to see that an extraction path models a conjunctive query of unary and binary conditions on a labeled ordered tree ([6]). Actually, an extraction path is a special kind of extraction pattern of arity 1 according to [3].

Figure 4a shows an extraction path. The extraction vertex is marked with a small arrow (vertex C in that figure). A node is extracted by this path if it has the following properties: i) it has two preceding left siblings; ii) it has one following right sibling that is the last child of its parent node; iii) it has a parent labeled with a ; iv) its parent has a following right sibling that is the last child of its parent node; v) its parent has a preceding left sibling that is the first child of its parent node; vi) it has a grand-parent; vii) it has a grand-grand-parent labeled with a that is the unique child of its parent node.

Consider an extraction path $p = [t_0, t_1, \dots, t_k]$. For a list $t_i = [v_{-l}, \dots, v_{-1}, v_0, v_1, \dots, v_r]$ let $left(t_i) = l$ and $right(t_i) = r$. The following definition introduces height, together with left and right widths of an extraction path.

Definition 2. (Height and widths of an extraction path) Let $p = [t_0, t_1, \dots, t_k]$ be an extraction path.

1. The value $height(p) = k$ is called the height of p .
2. The value $left(p) = \max_{i=0}^k left(t_i)$ is called the left width of p . The value $right(p) = \max_{i=0}^k right(t_i)$ is called the right width of p .

In practice it is useful to limit the height and the widths of an extraction path, yielding a *bounded extraction path*.

Definition 3. (*Bounded extraction paths*) Let H, L, R be three positive integers. An extraction path $p = [t_0, t_1, \dots, t_k]$ is called (H, L, R) -bounded if $height(p) \leq H$, $left(p) \leq L$ and $right(p) \leq R$.

Note that the extraction path shown in figure 4a is a $(3, 2, 1)$ -bounded extraction path. Moreover, if we restrict $H = 2$ and $L = 1$, then nodes I and A will be pruned resulting a $(2, 1, 1)$ -bounded extraction path, that obviously, is less constrained than the initial path.

3 A Path Generalization Algorithm

The practice of Web publishing assumes dynamically filling-in HTML templates with structured data taken from relational databases. Thus, we can safely assume that a lot of Web data is contained in sets of documents that share similar structures. Examples of such documents are: search engines result pages, product catalogues, news sites, product information sheets, travel resources, etc.

We consider a Web data extraction scenario which assumes the manual execution of a few extraction tasks by the human user. An inductive learning engine could then use the extracted examples to learn a general extraction rule that can be further applied to the current or other similar Web pages.

Usually the extraction task is focused on extracting similar items (for example book titles in a library catalogue, or product features in a product information sheet). One approach to generate an extraction rule from a set of manually extracted items is to discover a common pattern of their neighboring nodes in the tree of the target document.

In what follows we discuss an algorithm that takes: i) an XML document (possibly assembled from more Web pages, previously converted to XHTML) modeled as a labeled ordered tree t ; ii) a set of example nodes $\{e_1, e_2, \dots, e_n\}$; iii) three positive integers H, L, R and produces an (H, L, R) -bounded extraction path p that generalizes the set of input examples. Intuitively, this technique is guaranteed to work if we assume that semantically similar items will exhibit structural similarities in the target Web document. This is a feasible assumption for the case of Web documents that are generated on-the-fly by filling-in HTML templates with data taken from databases. Moreover, based on experimental results that we have recorded in previous work ([1, 2]), we have noticed that in practice an extraction rule only needs to check the proximity of nodes. This explains why we focused on the task of learning bounded extraction paths.

The basic operation of the learning algorithm is the generalization operator of two extraction paths. This operator takes two extraction paths p_1 and p_2 and produces an appropriate extraction path p that generalizes p_1 and p_2 .

The idea of the learning algorithm is as follows. For each example node we generate a bounded extraction path (of given input parameters H, L, R) by following sibling and parent links in the document tree. We initialize the output path with the first extraction path and then we proceed by iterative application of the generalization operator to the current output path and the next example extraction path, yielding a new output path. The result is a bounded extraction path that represents an appropriate generalization of the input examples.

The generalization of two paths assumes the generalization of their elements, starting with the elements containing the extraction vertices and moving upper level by level. The generalization of two levels assumes the generalization of each pair of corresponding vertices, starting with vertices with index 0 and moving to left and respectively to right in the lists of vertices. Generalization of two vertices is as simple as taking the intersection of their labels. The path generalization algorithm is shown in figure 2.

| | |
|---|--|
| <pre> LEARN(p_1, \dots, p_n, n) 1. $p \leftarrow p_1$ 2. for $i = 2, n$ do 3. $p \leftarrow \text{GEN-PATH}(p, p_i)$ 4. return p </pre> | <pre> GEN-PATH(p_1, p_2) 1. let $p_1 = [t_0^1, \dots, t_{k_1}^1]$ 2. let $p_2 = [t_0^2, \dots, t_{k_2}^2]$ 3. $k \leftarrow \min(k_1, k_2)$ 4. $i \leftarrow k, i_1 \leftarrow k_1, i_2 \leftarrow k_2$ 5. while $i \geq 0$ do 6. $t_i \leftarrow \text{GEN-LEVEL}(t_{i_1}^1, t_{i_2}^2)$ 7. $i \leftarrow i - 1, i_1 \leftarrow i_1 - 1, i_2 \leftarrow i_2 - 1$ 8. return $p = [t_0, \dots, t_k]$ </pre> |
| <pre> GEN-LEVEL(t_1, t_2) 1. let $t_1 = [v_{l_1}^1, \dots, v_{-1}^1, v_0^1, \dots, v_{r_1}^1]$ 2. let $t_2 = [v_{l_2}^2, \dots, v_{-1}^2, v_0^2, \dots, v_{r_2}^2]$ 3. $l \leftarrow \min(l_1, l_2)$ 4. $r \leftarrow \min(r_1, r_2)$ 5. for $i = 0, r$ do 6. $v_i \leftarrow \text{GEN-VERTEX}(v_i^1, v_i^2)$ 7. for $i = 1, l$ do 8. $v_{-i} \leftarrow \text{GEN-VERTEX}(v_{-i}^1, v_{-i}^2)$ 9. return $t = [v_{-l}, \dots, v_{-1}, v_0, \dots, v_r]$ </pre> | <pre> GEN-VERTEX(v_1, v_2) 1. let λ_1 be the label of v_1 2. let λ_2 be the label of v_2 3. $\lambda \leftarrow \lambda_1 \cap \lambda_2$ 4. return node v with label λ </pre> |

Fig. 2. Path generalization algorithm

Function LEARN generalizes the extraction paths of the example nodes. We assume that paths p_1, \dots, p_n are generated as bounded extraction paths before function LEARN is called. Function GEN-PATH takes two extraction paths p_1, p_2 and computes the generalized path p . Function GEN-LEVEL takes two lists of vertices t_1 and t_2 that are members of the extractions paths and computes a generalized list t that is member of the generalized path. Function GEN-VERTEX takes two vertices v_1, v_2 and computes a generalized vertex v .

It is easy to see that the execution of algorithm LEARN takes time $O(n \times H \times (L + R))$ because GEN-VERTEX takes time $O(1)$, GEN-LEVEL takes time $O(L + R)$ and GEN-PATH takes time $O(H \times (L + R))$. Note also that if we set $H = L = R = \infty$ then the complexity of the algorithm is $O(n \times H^* \times W^*)$ where H^* and W^* are the height and the width of the target document tree.

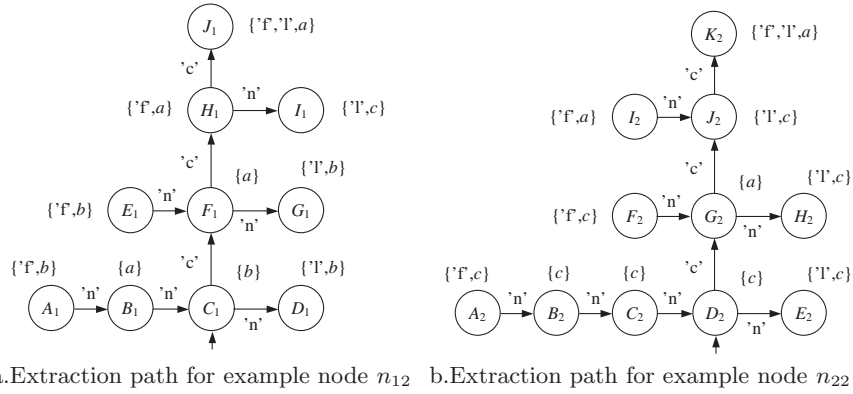


Fig. 3. Extraction paths for example nodes from figure 1

Consider again the labeled ordered tree shown in figure 1 and the example nodes marked with dashed rectangles (n_{12} and n_{22}). The extraction paths corresponding to these nodes are shown in figure 3. The result of applying the generalization algorithm on those paths is shown in figure 4a.

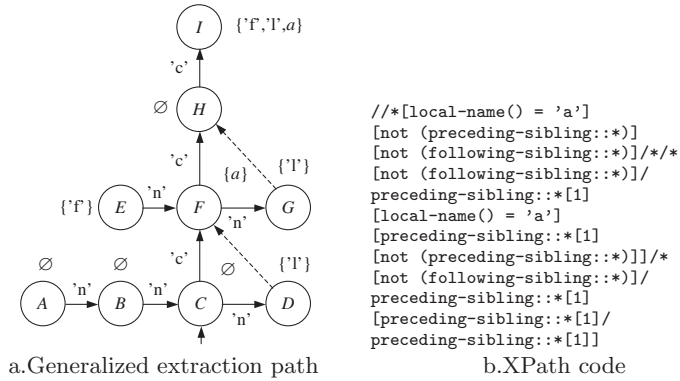


Fig. 4. Generalized extraction path for example from figure 1

4 Translating Extraction Paths to XSLT

An extraction path can be translated to an XPath query. The XPath query can be embedded into an XSLT stylesheet ([5]) to finally extract the information and store it into a database or another structured document ([3]).

Figure 5 shows an algorithm for translating an extraction path into an XPath query. The translation algorithm takes an extraction path $p = [t_0, \dots, t_k]$ and explores it starting with t_0 and moving to t_k . For each element t_i , $0 \leq i \leq k$, the

algorithm maps t_i to a piece of the output XPath query. Actually the algorithm takes the following route of vertices: $v_{r_0}^0 \rightarrow \dots \rightarrow v_0^0 \rightarrow v_{r_1}^1 \rightarrow \dots \rightarrow v_0^1 \rightarrow \dots \rightarrow v_{r_k}^k \rightarrow \dots \rightarrow v_0^k$. Note that when moving from element i to element $i+1$, $0 \leq i < k$, the algorithm takes the route $v_0^i \rightarrow v_{r_{i+1}}^{i+1}$ (opposite direction of dotted arrows in figure 4a) rather than the route $v_0^i \rightarrow v_0^{i+1}$. For each vertex v_0^i , $0 \leq i \leq k$, the algorithm also generates a condition that accounts for their left siblings by taking the route $v_0^i \rightarrow v_{-1}^i \rightarrow \dots \rightarrow v_{-i}^i$. It is easy to see that if p is an (H, L, R) -bounded extraction path then the time complexity of the translation algorithm PATH-TO-XPATH is $O(H \times (L + R))$.

Fig 4b shows the result of applying this algorithm to the extraction path from figure 4a. The algorithm will explore the following route of vertices: $I \rightarrow H \rightarrow G \rightarrow F \rightarrow D \rightarrow C$. For each vertex the algorithm generates a location step comprising an axis specifier, a node test and a sequence of predicates written between '[' and ']'. The node test is always $*$. The axis specifier is determined by the relation of the current vertex with its preceding vertex on the route explored by the translation algorithm. For example, the axis specifier that is generated for vertex F is `preceding-sibling::`. In this later case, an additional predicate `[1]` that constraints the selection of exactly the preceding node, is added. The algorithm also generates a predicate for each element of the label of a vertex. For example, predicate `[local-name() = 'a']` is generated for vertex F , that checks the node tag, and predicate `[not (following-sibling::*)]` is generated for vertex G , that checks if the matched node is the last child of its parent node. Moreover, for vertices F and C the algorithm generates an additional predicate that accounts for their left siblings E (of F) and respectively $B \rightarrow A$ (of C). For example, additional predicate `[preceding-sibling::*[1][not (preceding-sibling::*)]]` is generated for vertex F . This predicate checks if the document node matched by vertex F has a predecessor and if the predecessor is the first child of its parent node.

Note that running the XPath query from figure 4b on the labeled ordered tree from figure 1 produces the following two answers `/a[1]/a[1]/a[1]/b[3]` and `/a[1]/c[1]/a[1]/c[4]` that correspond to nodes n_{12} and n_{22} .

5 Experiment

We performed a simple (but realistic) experiment using data harvested from the Expedia Web site with the task of extracting hotel names. We followed a process consisting of the following stages: page collection, pre-processing, manual information extraction, conversion to the input format of the learning program, learning, wrapper compilation, wrapper execution ([2]).

We selected a sample page set of 50 pages containing all the results we got by searching Expedia for hotels in Paris (an example is shown in figure 6). This set contained 1248 hotels, 25 hotels per page (excepting last page with only 23 hotels). We converted each page to XHTML using the Tidy program and then we assembled all these files into a single XML file by concatenating them under a new root element. The resulting file had about 29 Mb comprising a total of

```

PATH-TO-XPATH( $p$ )
1. let  $p = [t_0, \dots, t_k]$ 
2.  $xp \leftarrow ""$ 
3. for  $i = 0, k$  do
4.   let  $t_i = [v_{-l}, \dots, v_{-1}, v_0, \dots, v_r]$ 
5.    $t \leftarrow t_i$ 
6.    $xp \leftarrow xp + "/" + \text{COND}(v_r)$ 
7.   for  $j = r - 1, 0$  do
8.      $xp \leftarrow xp + "/\text{preceding-sibling::*[1]}" +$ 
8.      $\text{COND}(v_j)$ 
9.   if  $l > 0$  then
10.     $xp \leftarrow xp + "/\text{preceding-sibling::*[1]}" +$ 
11.     $\text{COND}(v_{-1})$ 
11.    for  $j = 2, l$  do
12.       $xp \leftarrow xp + "/\text{preceding-sibling::*[1]}" +$ 
13.       $\text{COND}(v_{-j})$ 
13.     $xp \leftarrow xp + "]"$ 
14. return  $xp$ 

COND( $v$ )
1. let  $\lambda$  be the label of  $v$ 
2.  $xc \leftarrow ""$ 
3. if there is  $\sigma \in \lambda \cap \Sigma$  then
4.    $xc \leftarrow xc +$ 
5.    $"[\text{local-name()}=\sigma]"$ 
5. if there is 'f'  $\in \lambda \cap \Sigma$  then
6.    $xc \leftarrow xc +$ 
7.    $"[\text{not} (\text{preceding-sibling::*})]"$ 
7. if there is 'l'  $\in \lambda \cap \Sigma$  then
8.    $xc \leftarrow xc +$ 
9.    $"[\text{not} (\text{following-sibling::*})]"$ 
9. return  $xc$ 

```

Fig. 5. Algorithm for translating an extraction path into XPath

191816 nodes. Note that the size of this training document is about two orders of magnitude larger than the one used in [1].

Page 1 of 50 Previous | **Next**

Sort by: **Expedia Picks** Price Hotel Name City Hotel Class

K and K Hotel Cayre St Germain des Pres Lowest avg rate **\$287.55**

Paris, France Area: Eiffel Tower-Orsay Museum (7)

Stylish boutique hotel on the Left Bank
 This seven-story hotel is located in the Left Bank neighborhood of St-Germain-des-Prés, two blocks from the Musée d'Orsay and 500 meters (a quarter-mile) from ...
[More lodging info](#)

Availability request: 1 room Expedia Special Rate Mon Apr-3-2006 to Fri Apr-7-2006

| Room type | Mon | Tue | Wed | Thu | Avg rate (per night) | |
|--|-------|-------|-------|-------|----------------------|-------------------------|
| Classic Single room | \$288 | \$288 | \$288 | \$288 | \$287.55 | Book it |
| Classic Double room | \$315 | \$315 | \$315 | \$315 | \$314.59 | Book it |
| Executive Double Room-Executive Double | \$361 | \$361 | \$361 | \$361 | \$361.28 | Book it |
| Classic Triple Room | \$378 | \$378 | \$378 | \$378 | \$378.49 | Book it |

Le Regent Montmartre Lowest avg rate **\$98.31**

Paris, France Area: L'Opera (9)

- Next to Anvers metro, at the foot of the Butte Montmartre, this budget hotel captures the charm of Montmartre's bohemia.
- Rooms are ...

[More lodging info](#)

Fig. 6. Sample page resulted by searching hotels on Expedia

We run the learner on this sample file for 5 training examples that were randomly selected from result pages 1, 5, 11, 24 and 46 and parameters values set to $H = 5$ and $L = R = 3$. The resulted extraction path was converted to XPath and then the XPath query was run on the initial document and on other 80 result pages, obtained by searching other hotels on Expedia. As result, excellent values were recorded for both precision and recall – both values were 1. The resulting wrapper expressed in XSLT is shown in the appendix.

6 Related Works

The rapid expansion of the Web attracted a lot of researches in the area of information extraction and wrapper induction. Overviews of related technologies and systems can be found in: [3], [7], and [8]. We have chosen for discussion in this section some works that we think that are closer to what has been presented in this paper, namely [1–3], [9], and [4].

A special class of wrappers called L-wrappers (i.e. *logic wrappers*) for tuples extraction, that were inspired by the logic programming paradigm, is studied in papers [1–3]. L-wrappers i) have a declarative semantics, and therefore their specification is decoupled from their implementation and ii) can be generated using inductive logic programming. The extraction paths introduced in the current paper are just a special class of L-wrappers of arity 1 for which we devised a new and more efficient learning algorithm that runs in polynomial time.

In paper [9] tree wrappers for tuples extraction are introduced. A tree wrapper is a sequence of tree extraction paths. There is an extraction path for each extracted attribute. A tree extraction path is a sequence of triples that contain a tag, a position and a set of tag attributes. A triple matches a node based on the node tag, its position among its siblings with a similar tag and its attributes. Extracted items are assembled into tuples by analyzing their relative document order. The algorithm for learning a tree extraction path is quite similar to ours – the composition of two tree extraction paths corresponds to the generalization operator of two extraction paths (our GEN-PATH algorithm). Note also that our extraction paths use a different and richer representation of node proximity and therefore, we have reasons to believe that our wrappers could be more accurate (this claim needs, of course, further support with experimental evidence). Finally, note that our approach can also be extended to tuples extraction as in [9]. However, as future work we are interested to extend the approach presented in our paper to devise an efficient learner for L-wrappers for tuples extraction ([2, 3]), rather than following the approach outlined in [9].

Finally, a generalization of the notion of string delimiters developed for information extraction from string documents ([7]) to subtree delimiters from tree documents is described in paper [4]. This paper introduces a special purpose learner that constructs a structure called candidate index based on trie data structures. Note that the tree leaf delimiters described in this paper are quite similar to our extraction paths and the representation of reverse paths using the symbols $Up(\uparrow)$, $Left(\leftarrow)$ and $Right(\rightarrow)$ can be easily simulated in our approach by 'c' and 'n' arcs.

7 Concluding Remarks

In this note we presented an efficient algorithm for learning to extract single information items from HTML documents. The algorithm is based on the idea of generalizing extraction paths. As future work we intend to investigate the extension of this algorithm to learn L-wrappers for tuples extraction.

References

1. Bădică, C., Bădică, A.: Rule Learning for Feature Values Extraction from HTML Product Information Sheets. In: Boley, H., Antoniou, G. (eds): *Proc. RuleML'04*, Hiroshima, Japan. LNCS 3323 Springer-Verlag (2004) 37–48.
2. Bădică, C., Bădică, A., Popescu, E.: Tuples Extraction from HTML Using Logic Wrappers and Inductive Logic Programming. In: Szczepaniak, P.S., Kacprzyk, J., Niewiadomski, A. (eds.): *Proc. AWIC'05*, Lodz, Poland. LNAI 3528 Springer-Verlag (2005) 44–50.
3. Bădică, C., Bădică, A.: Logic Wrappers and XSLT Transformations for Tuples Extraction from HTML. In: Bressan, S.; Ceri, S.; Hunt, E.; Ives, Z.G.; Bellahsene, Z.; Rys, M.; Unland, R. (eds): *Proc. 3rd International XML Database Symposium XSym'05*, Trondheim, Norway. LNCS 3671, Springer-Verlag (2005) 177–191
4. Chidlovskii, B.: Information Extraction from Tree Documents by Learning Subtree Delimiters. In: *Proc. IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico (2003) 3–8.
5. Clark, J.: XSLT Transformation (XSLT) Version 1.0, W3C Recommendation, 16 November 1999, <http://www.w3.org/TR/xslt> (1999).
6. Gottlob, G., Koch, C., Schulz, K.U.: Conjunctive Queries over Trees. In: *Proc. PODS'2004*, Paris, France. ACM Press, (2004) 189–200.
7. Kushmerick, N., Thomas, B.: Adaptive Information Extraction: Core Technologies for Information Agents, In: *Intelligent Information Agents R&D in Europe: An AgentLink perspective* (Klusch, Bergamaschi, Edwards & Petta, eds.). LNCS 2586, Springer-Verlag (2003).
8. Li, Z., Ng, W.K.: WDEE: Web Data Extraction by Example. In: L. Zhou, B.C. Ooi, and X. Meng (Eds.): *Proc. DASFAA'2005*, Beijing, China. LNCS 3453, Springer-Verlag (2005), 347–358.
9. Sakamoto, H., Arimura, H., Arikawa, S.: Knowledge Discovery from Semistructured Texts. In: Arikawa, S., Shinohara, A. (eds.): *Progress in Discovery Science*. LNCS 2281, Springer-Verlag (2002) 586–599.
10. World Wide Web Consortium. XML Path Language (XPath) Recommendation. <http://www.w3c.org/TR/xpath/>, November 1999.

A XSLT Code of the Sample Wrapper

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="html">
    <result>
      <xsl:apply-templates mode="selhotel" select=
        "//*[not (preceding-sibling:*)][not (following-sibling:*)]/
        tr[not (preceding-sibling:*)][not (following-sibling:*)]/
        td[preceding-sibling::*[1][local-name() = 'td']][not (preceding-sibling:*)]/
        a[not (preceding-sibling:*)][not (following-sibling:*)]/
        font[not (preceding-sibling:*)][not (following-sibling:*)]/
        text()[not (preceding-sibling:*)][not (following-sibling:*)]" />
    </result>
  </xsl:template>
  <xsl:template match="node()" mode="selhotel">
    <xsl:variable name="var_hotel"> <xsl:value-of select="normalize-space(.)" />
  </xsl:variable>
  <hotel><xsl:attribute name="hotel_name"> <xsl:value-of select="$var_hotel" />
  </xsl:attribute></hotel>
  </xsl:template>
</xsl:stylesheet>

```