Interactive Tuples Extraction from Semi-Structured Data

Rémi Gilleron Patrick Marty Marc Tommasi
Fabien Torre
INRIA Futurs and Lille University; email: first.last@univ-lille3.fr

Abstract

This paper studies from a machine learning viewpoint the problem of extracting tuples of a target n-ary relation from tree structured data like XML or XHTML documents. Our system can extract, without any post-processing, tuples for all data structures including nested, rotated and cross tables. The wrapper induction algorithm we propose is based on two main ideas. It is incremental: partial tuples are extracted by increasing length. It is based on a representation-enrichment procedure: partial tuples of length i are encoded with the knowledge of extracted tuples of length $i-1$. The algorithm is then set in a friendly interactive wrapper induction system for Web documents. We evaluate our system on several information extraction tasks over corporate Web sites. It achieves state-of-the-art results on simple data structures and succeeds on complex data structures where previous approaches fail. Experiments also show that our interactive framework significantly reduces the number of user interactions needed to build a wrapper.

1 Introduction

In less than ten years, XML has become a standard for semi-structured data exchange and storage. From its preliminary usages in Internet technologies, XML influence has grown to become widely used now for corporate data[1]. XML usage includes document-like data produced for instance by word processors and spreadsheets, data for software interoperability (UDDI, WSDL). In the meantime, with the generalization of decision support systems, high valued corporate data such as economic statistics are generated by reporting tools[2] and are published as semi-structured documents. All such data participates in a new source of information that end-users want to query in an easy way.

But end-user accessibility of XML or XHTML data, by means of query tools, is more problematic than in the classical and long-standing experienced case of relational databases. This situation is due either to the lack of explicit and coercive data types or due to the extensibility nature of XML that allows great heterogeneity of data.

Therefore, in spite of standardized query languages like XPATH, a common attitude is to develop specific programs for each query over semi-structured data. Recently, techniques have appeared to assist end-users in designing such programs. For instance, the Lixto system [2] allows to define queries or so-called *wrappers* over semi-structured data in a visual and interactive framework. Unfortunately, specifying wrappers with such systems can be too difficult for end-users. Alternate or complementary approaches relying on machine learning techniques have been considered. As in a query by example system, an end-user annotates data to extract and the system induces an appropriate wrapper. Machine learning approaches have been widely developed for Web information extraction [3, 5, 9, 11, 14, 16, 17]. These systems are limited so far to simple Web pages containing lists of results *à la Google* or tables.

But data organizations in tree structured documents may be intricate. For instance, let us consider as a running example a (part of a) page of the BEA Web site from www.bea.gov in Fig. 1. The corresponding HTML tree is given in Fig. 2. Let us consider the target relation (Country, Year, Exports, Balance). The tuples (France, 1986, 10.130, 7.119) and (France, 1987, 11.701, 7.947) are intertwined in the HTML tree. Thus extracting tuples in the document order should fail. Also it could be noted that values for the component Country are factorized over several tuples.

Our objective is a friendly system for extracting tuples from XML or XHTML documents whatever is the structure used to store tuples. We follow the wrapper induction approach which is licensed because the generation process implies regularities of semi-structured data. We first study, in Section 2, different ways to encode n-ary relations in XML, that is the manner to store tuples in XML trees. We define five base cases: lists, tables, rotated tables, nested tables and cross tables. Related work is described in Section 3. It is shown that existing systems for extracting tuples from Web documents only deal with lists and tables. Other systems

[1]See for instance www.oasis-open.org
[2]See for instance jasperreports.sourceforge.net

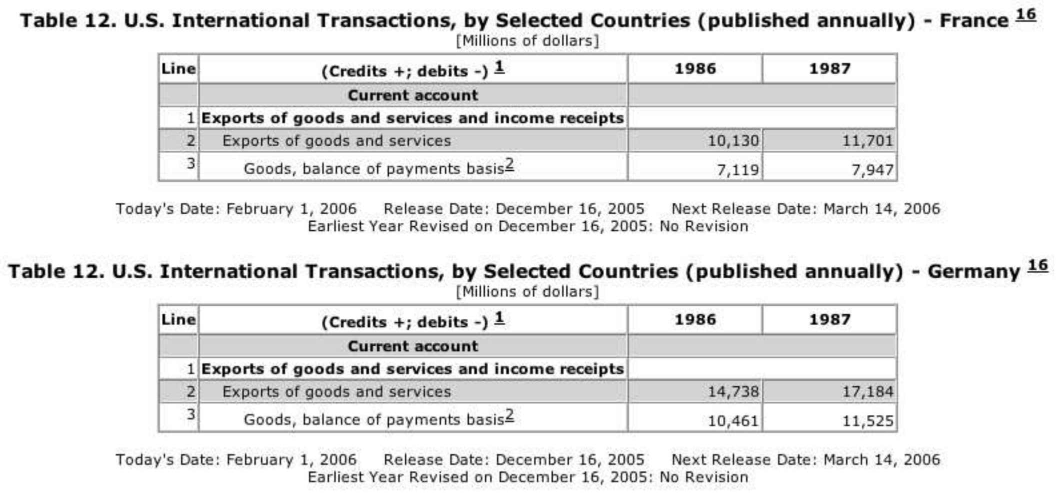


Table 12. U.S. International Transactions, by Selected Countries (published annually) - France 16

[Millions of dollars]

Line	(Credits +; debits -) 1	1986	1987
	Current account		
1	**Exports of goods and services and income receipts**		
2	Exports of goods and services	10,130	11,701
3	Goods, balance of payments basis2	7,119	7,947

Today's Date: February 1, 2006 Release Date: December 16, 2005 Next Release Date: March 14, 2006
Earliest Year Revised on December 16, 2005: No Revision

Table 12. U.S. International Transactions, by Selected Countries (published annually) - Germany 16

[Millions of dollars]

Line	(Credits +; debits -) 1	1986	1987
	Current account		
1	**Exports of goods and services and income receipts**		
2	Exports of goods and services	14,738	17,184
3	Goods, balance of payments basis2	10,461	11,525

Today's Date: February 1, 2006 Release Date: December 16, 2005 Next Release Date: March 14, 2006
Earliest Year Revised on December 16, 2005: No Revision

Figure 1. Part of a sample page of BEA web site.

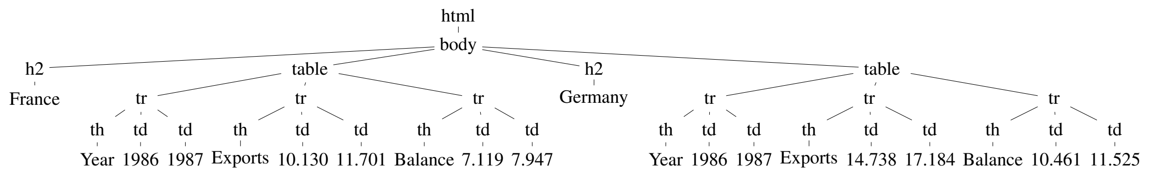


Figure 2. Simplified HTML tree for a page of the BEA Web site (Fig. 1). It corresponds to rotated tables with pivot values

rely on unary wrappers and a post processing to enable tuple extraction. But let us consider the BEA example and the target relation (Country, Year, Exports, Balance) and let us suppose that the sets of values for each component of the target relation are known, the post processing needs the knowledge of the tree structure depicted in Fig. 2. Writing such a post processing procedure is not easy for an end-user. Moreover, writing by hand such a procedure is in conflict with the example-based approach.

Thus a machine learning approach for extracting tuples should allow to extract tuples whatever is the data organization. The number of tuples in a tree is exponential in the length of tuples. Therefore, direct approaches fail and we propose a system based on two main ideas. It is *incremental*: partial tuples are extracted by increasing length. It is based on a *representation-enrichment procedure*: partial tuples of length i are encoded with the knowledge of extracted tuples of length $i-1$. In Section 4, we present the extraction algorithm. At each step of the incremental process, a binary classification procedure is used. The attribute-value representation schemas use both the DOM view (tree structure) and the yield view (textual contents) of XML data. The induction procedure from completely annotated documents, in which all tuples to be extracted are annotated, is given in Section 5. At each step i of the incremental process, a classifier is learned from examples. Positive examples are encodings of partial tuples which are projections of tuples to be extracted. Negative examples are encodings of partial tuples not to be extracted while the $i-1$-th partial tuple is to be extracted. This ensures that the number of examples is linear in the number of annotated tuples.

The system suffers of one main flaw: completely annotated examples are required. As our objective is to build a friendly end-user system for wrapper generation with machine learning techniques, it is mandatory to be able to build accurate wrappers in a minimal amount of time. In Section 6, we integrate our approach in an interactive framework in order to reduce the number of user actions. This implies to modify the algorithms to learn from partially annotated examples, in which some tuples to be extracted are annotated as positive examples and some tuples not to be extracted are annotated as negative.

In Section 7, we present experimental results of our prototype. It has been evaluated on standard RISE [18] data sets and corporate Web sites in the domains of statistics, meteorology and economics. These data sets cover the simplest organizations (lists and tables) and more complex ones (rotated tables, nested tables and cross tables). First, we consider the case of completely annotated examples. Our system achieves excellent results without any post-processing. On simple datasets it reaches state-of-the-art results and it succeeds on intricate ones where other systems fail. Second, we study the behaviour of our algorithm in an interactive framework. We show that this interactive approach significantly improves the time needed to build wrappers when the number of tuples in each document is large. It also reduces the quantity of information provided by the user to define accurate wrappers.

2 Tuples in XML data

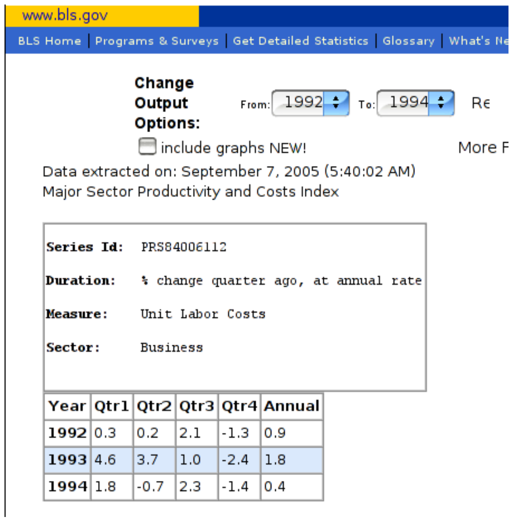
We discuss how tuples can be embedded in XML data. Our case study was done for XHTML data from different Web sites. We present the five base cases:

Case 1. Tuples can be stored consecutively in the same order. For instance consider a list of search results like Google or eBay. Tuples do not overlap in the XHTML document. Two cases should be distinguished. In the first one, the tree structure of the XHTML document is poor and the problem is equivalent to a problem for sequences. In the second one, the list constructor of XHTML is used. In this case, each tuple is in a different item. If nested lists are used, the case of lists is quite similar to the case of tables we present now.

Case 2. A second structure is a table with component names in the first row and data in the following rows[3]. Tuples again do not overlap in the XHTML document. In the DOM view the least common ancestors of each tuple are distinct. So it is easy to distinguish between tuples using the tree structure or the XHTML tags in the document. Nested lists lead to similar properties.

Case 3. For relations with more components than data, the previous organization is often rotated. Slot names are in the first column and data are in the following columns[4]. Another example is given by the tables from www.bea.gov in Fig. 1 for the target relation (Year, Exports, Balance) for France. It is shown in Fig. 2 that tuples (1986, 10.130,7.119) and (1987, 11.701, 7.947) are intertwined. It should also be noted that different components of a tuple occur at the same position in the DOM tree. For instance, 1987, 11.701 and 7.947 are below the third td of the tr tags of the table.

Case 4. A nested structure is used to avoid repetitions and to shorten the data presentation. This is common when issued from reporting tools. Consider for instance the address report from JASPERREPORTS [5] where the city name is factorized among several citizens records. Another example is given by the tables from www.bea.gov in Fig. 1 for the target relation (Country, Year, Exports, Balance). The country name is factorized and it is shown in Fig. 2 that component values occur at different depths in the DOM tree.


www.bls.gov
BLS Home | Programs & Surveys | Get Detailed Statistics | Glossary | What's Ne
Change Output Options:
From: 1992
To: 1994
Re
include graphs NEW!
More F

Data extracted on: September 7, 2005 (5:40:02 AM)
Major Sector Productivity and Costs Index

Series Id:	PRS84006112
Duration:	% change quarter ago, at annual rate
Measure:	Unit Labor Costs
Sector:	Business

Year	Qtr1	Qtr2	Qtr3	Qtr4	Annual
1992	0.3	0.2	2.1	-1.3	0.9
1993	4.6	3.7	1.0	-2.4	1.8
1994	1.8	-0.7	2.3	-1.4	0.4

Figure 3. Part of a sample page of BLS web site.

Case 5. Cross-tables are used to present multi-dimensional relations. This is frequent for statistics and reports. This case shares properties with the two preceding cases: a subset of the component values are stored as in the case 3 but the rest is stored according to case 4. An example from BLS www.bls.gov is given in Fig. 3. The DOM tree is given in Fig. 4. Let us consider for instance the target relation (Year, Quarter, Unit Labor Costs). Examples of tuples are (1992, Qtr1, 0.3) and (1993, Qtr3, 1.0).

For the two first cases, tuples are stored consecutively while, for the three last cases, tuples are intertwined both in the sequential view and in the DOM view. For the three last cases, it should be necessary to count in order to construct tuples because component values occur at the same position in different subtrees. It should be noted that variants are frequent in XML or XHTML data. They consist in mixing several possible layouts, merging some leaves of the tree or placing several relations in the same structure. Also, building tuples can be more intricate when there are missing values in tuples.

3 Related work

We relate to [12] for a survey on Web data extraction tools. Here we focus on wrapper induction tools for Web documents. We discuss the supervised and unsupervised

[3] see the table of market indices on quotes.nasdaq.com/aspx/marketindices.aspx
[4] see the table of http://www.bls.gov/eag/eag.IA.htm
[5] jasperreports.sourceforge.net/samples/DataSourceReport.html

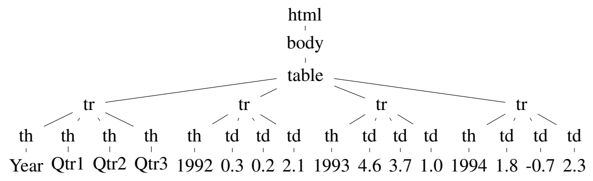

html
body
table
tr
tr
tr
tr
th th th th th td td td th td td td th td td td
Year Qtr1 Qtr2 Qtr3 1992 0.3 0.2 2.1 1993 4.6 3.7 1.0 1994 1.8 -0.7 2.3

Figure 4. Tree organization for a cross-table from BLS Web site presented in Fig. 3 restricted to the three first quarters.

approaches.

The supervised wrapper induction problem has been addressed by many authors. Several systems have emerged from this active research field, working from manually annotated documents. Let us cite WIEN [11], SOFT MEALY [9], BWI [7], STALKER [14], SQUIRREL [3], among others. They achieve very good performance on unary extraction tasks. Wrapper induction approaches for n-ary tuples extraction include Kushmerick seminal works [11] with LR wrappers and transducers induction in [9]. More recently Thomas proposes an approach based on Inductive Logic Programming [17]. These systems are designed for HTML documents generated from a back-end database, such as commercial sites and online shops. For instance, LR wrappers work when tuples are stored consecutively in the sequential view of HTML documents. Thus, to the best of our knowledge, wrapper induction systems for n-ary relations are able to deal with the most simple cases: lists, nested lists and tables.

For n-ary extraction, it is often argued that a post-processing that builds tuples from components extracted by n unary wrappers is sufficient to solve the n-ary extraction problem. Indeed, this post processing is easy for cases 1 or 2 when there are no missing values. With missing values, user interactions are required to label data or to correctly re-associate tuple components [5, 10]. We argue that this is unsatisfactory for at least two reasons. First, we have shown in the introduction that, even with perfect unary wrappers, the post-processing procedure may be complex. Second, learning-based solutions have been promoted to avoid writing wrappers by hand. Thus it is not fair to ask the end user to write programs for constructing tuples from the outputs of several unary wrappers.

The unsupervised approach is automatically trying to avoid manual labeling. Unsupervised learning methods are based on document alignment [1, 4, 13, 20] or grammatical inference techniques [6]. However, as indicated by Zhai and Liu in [19], these automatic methods are less accurate than the systems that ask the user to label training pages. Manual post-processing is needed for the user to identify what he/she is interested in. For instance, consider the tables from www.bea.gov in Fig. 1, the system presented in [20] identifies the two data regions corresponding to the two tables. But if we consider the target relation (Country, Year, Exports, Balance), the problem of extracting tuples is still present. Moreover, in case 4, factorized values may not be found by an unsupervised system. Therefore, extracting tuples of records from the output of automatic systems can be as difficult as wrapping original documents. Nevertheless, automatic systems could be a useful preprocessing for large documents with few data records in order to identify the data regions.

To sum-up, existing systems based on machine learning techniques solve simple data organizations (Cases 1 and 2), but fail on complex data organizations (Cases 3-5).

4 The extraction process

We present the incremental extraction procedure which is based on data enrichment. The first component of the target n-ary relation is called the *seed*. The seed extraction is cast as a binary classification problem, which consists in deciding whether a leaf is to be extracted or not. Figure 5 briefly describes the 56 attributes used by rep_1 to encode a leaf. The reader should note that attributes are defined from several views over XML or XHTML documents: nodes properties over the *DOM view*; textual values over the *yield view* which is the concatenation of text leaves contents obtained with a depth-first left-right exploration of the DOM view.

Extraction of partial tuples of length i ($i \neq 1$) is cast as a binary classification problem, which consists in deciding whether a tuple of length i should be extracted or not. Tuples of length i are encoded in an attribute-value representation denoted by rep_i. Let us consider a tuple $(l_0, \ldots, l_{i-1}, l_i)$, rep_i includes an encoding of dependencies between l_i and the partial tuple $(l_0, \ldots, l_{i-1})$. For encoding dependencies, we use couple node attributes which are presented in Fig. 6. Again couple node attributes are defined over the DOM view and the yield view. In order to have a fixed number of attributes whatever the length of the partial tuple is, rep_i encodes a partial tuple $(l_0, \ldots, l_{i-1}, l_i)$

Node representation label, position in father's children sequence, depth and height, number of children, size of the subtree under the node, the label of its previous and next siblings, the value of the class XHTML attribute. There are 9 attributes.
Leaf representation node representation of the leaf plus node representation of its 5 ancestors in the DOM view plus the previous and next leaves contents (without tokenisation) in the yield view. There are 56 attributes.

Figure 5. Attributes for leaf representation.

couple nodes attributes: for a pair of nodes (p, m) attributes are: attributes for the node representation of the nearest common ancestor a between p and m; the length of the shortest path between p and a, between p and m and between a and m in the `DOM` view and in the binary tree encoding of the `DOM` view; the number of textual leaves between p and m in the `yield` view; and for each $1 \leq k \leq 5$; the difference between the position of the k^{th} ancestor of p and of the k^{th} ancestor of m in their father's children sequence. There are 21 attributes for each pair of nodes.

Figure 6. Attributes for pairs of nodes

by the leaf representation of l_i, couple node attributes for the pair (l_0, l_i) (dependency with the seed) and couple node attributes for the pair (l_{i-1}, l_i) (dependency with the previous component). There are 98 attributes for a tuple representation independently of i.

The extraction algorithm for a target n-ary relation is the algorithm 1. It takes as input a document d and n classifiers $c_1, \dots, c_n$ according to the representation schemas $rep_1, \dots, rep_n$. First, tuples of length 1 are extracted. Then, for each i, in line 4 a set S_i of candidate tuples to be extracted is defined by adding a leaf to every extracted tuple of length $i-1$. Tuples of length i are extracted in line 5: the representation procedure rep_i issues a record; records are then given to a classification procedure c_i that labels each of them as positive or negative. Every tuple of length i classified as positive is extracted. This allows us to handle factorized values. Indeed, when one of the $i-1$ components has factorized values, several tuples of length i, constructed with the same tuple t_{i-1} of length $i-1$, must be extracted. The line 6 allows us to handle missing values. When no tuple t_i, constructed from some t_{i-1}, is extracted, then the component i is assumed to be missing, and the tuple $(t_{i-1}, \texttt{null})$ is extracted.

The complexity of algorithm 1 depends on the complexity of computing rep_i for candidate partial tuples at each step i. The computation of tuples representation is efficient. Indeed, leaf and node attributes are computed while reading the input document. Couple nodes attributes must be computed at each step of the incremental extraction process. They are computed only from extracted tuples at the previous step. Couple nodes attributes involve finding the nearest common ancestor of a pair of nodes. This can be done in constant time after a linear preprocessing of the input tree structured document [8]. It should also be noted that a classifier c_i may only use a subset of attributes reducing the computation time for the extraction process.

Algorithm 1 Extraction algorithm

```
Input: document $d$; $n$ classifiers $c_1, \dots, c_n$ according to representation schemas $rep_1, \dots, rep_n$
Notation: $L(d)$ is the set of leaves of $d$, $S_i$ is the set of candidate partial tuples of length $i$, $S_i^+$ is the set of selected partial tuples of length $i$.
1: $S_1 = \{(l) \mid l \in L(d)\}$;
2: $S_1^+ = \{t_1 \in S_1 \mid c_1(rep_1(t_1)) = +1\}$
3: for $i = 2$ to $n$ do
4:     $S_i = \{t_i = (t_{i-1}, l) \mid t_{i-1} \in S_{i-1}^+,\ l \in L(d)\}$
5:     $S_i^+ = \{t_i \in S_i \mid c_i(rep_i(t_i)) = +1\}$
6:     $S_i^+ = S_i^+ \cup \{(t_{i-1}, \texttt{null}) \mid t_{i-1} \in S_{i-1}^+,\ \forall l \in L(d),\ c_i(rep_i((t_i, l))) = -1\}$
7: end for
Output: the set of extracted $n$-ary tuples $S_n^+$
```

5 The induction process

Let us consider a target n-ary relation. The learning process is incremental similarly to the extraction process. Given a set D of completely annotated documents, algorithm 2 loops from 1 to the number n of components. At each step i, it computes a classifier c_i for tuples of length i. The number of positive examples is bounded by the number of tuples in $\cup_{d \in S} S^+(d)$, that is the number of tuples to be extracted in D. On the opposite, the number of negative examples can be exponential in n. So we only use a subset of negative examples to train each classifier c_i. The selection of negative examples is driven by the schema of the extraction process as well: at each step i of the process, candidate tuples are built from positive examples at step $i-1$. Negative examples are computed by the function Neg taking as input a document d in D and a set of tuples to be extracted from d. It is defined by:

$$\mathsf{Neg}(d, S) = \{(x, l) \notin S | l \in L(d), \exists l' \in L(d) | (x, l') \in S\}$$

where $L(d)$ is the set of leaves of the document d. Then, examples are represented via rep_i and the base supervised classification algorithm W is applied. Algorithm 2 outputs a sequence of classifiers. This sequence will be the input of algorithm 1 for the extraction process.

6 Interactive wrapper induction system

Algorithm 2 suffers of one main flaw: it requires a set of documents where all tuples to extract are annotated. In this section, we propose to adapt this algorithm for learning from partially annotated examples in order to plug it in an interactive system. The aim is to reduce the number of user interactions to build n-ary wrappers.

In an interactive process, an end user first labels some examples to extract. Then we enter a loop where the system

Algorithm 2 Induction algorithm

```
Input: a sample $D = \cup\{d\}$ of annotated documents
Notation: $S^+(d)$ is the set of tuples to be extracted in $d$;
    $S_i^+(d)$ is the projection of $S^+(d)$ over the first $i$ components
1: for $i = 1$ to $n$ do
2:     $S_i^+ = \cup_{d \in D} S_i^+(d)$
3:     $S_i^- = \cup_{d \in D} \mathsf{Neg}(d, S_i^+(d))$
4:     $c_i = W(rep_i(S_i^+), rep_i(S_i^-))$
5: end for
Output: the sequence $(c_1, \ldots, c_n)$ of classifiers
```

proposes a new hypothesis wrapper and the user corrects this hypothesis until a good one is built. We have integrated the interactive process in the wrapper induction system following the incremental approach developed so far: for every i, wrappers for partial tuples are constructed in an interactive process.

We simulate an end user by an oracle U. The oracle U has given access to a pool of documents. Given as input an hypothesis wrapper $w_i = (c_1, \ldots, c_i)$ where each c_j is a classifier for partial tuples of length j, the oracle $U(w_i)$ returns true if w_i is correct and returns *new annotations* otherwise. We suppose a working document dc and that U returns new annotations according to the following scenario. If $U(w_i)$ does not return true and if w_i is not correct on dc, U returns a false negative example (a tuple not extracted by w_i while it should be extracted) and a false positive example (a tuple extracted by w_i while it should not be extracted) from dc. We should note that a false positive or a false negative may not exist. If $U(w_i)$ does not return true and w_i is correct on dc, then U returns a new document with two partial tuples of length i to be extracted.

The interactive wrapper induction system is the algorithm 3. It iterates over length of partial tuples. At each step i, it learns from a set of completely annotated documents D_i and a current working document dc, which is partially annotated. In the working document dc, $S_i^+(dc)$ (respectively $S_i^-(dc)$) denotes the set of partial tuples of length i to be extracted (respectively tuples not to be extracted) given by the oracle U.

When U returns a new document, it is supposed, according to the above scenario, that the current wrapper is correct over dc. Then, we add dc to the set of completely annotated documents and the new working document is the document returned by U.

The sets $S_i^+(dc)$ and $S_i^-(dc)$ are updated according to the annotations given by U.

Positive examples are partial tuples of length i to be extracted in the documents of D_i and in the set $S_i^+(dc)$.

Negative examples for completely annotated documents in D_i are defined according to function Neg as in Algo-

Algorithm 3 interactive wrapper induction system

```
Notation: an oracle $U$ is an end user simulator;
    $d(U(c_1, \ldots, c_i))$ is the document returned by $U$;
    $d^+(U(c_1, \ldots, c_i))$ is the set of tuples to be extracted returned by $U$;
    $d^-(U(c_1, \ldots, c_i))$ is the set of tuples not to be extracted returned by $U$;
1: for $i = 1$ to $n$ do
2:     $D_i = \emptyset$ {set of completely annotated documents}
3:     $c_i = \texttt{null}$ {classifier}
4:     $dc = \texttt{null}$ {current working document}
5:     while not $U(c_1, \ldots, c_i)$ do
6:         if $d(U(c_1, \ldots, c_i)) \neq dc$ then
7:             {a new document to be processed}
8:             $D_i = D_i \cup \{dc\}$ {$dc$ is completely annotated}
9:             $dc = d(U(c_1, \ldots, c_i))$; $S_i^+(dc) = S_i^-(dc) = \emptyset$
10:        end if
11:        $S_i^+(dc) = S_i^+(dc) \cup d^+(U(c_1, \ldots, c_i))$
12:        $S_i^-(dc) = S_i^-(dc) \cup d^-(U(c_1, \ldots, c_i))$
13:        $S_i^+ = (\cup_{d \in D_i} S_i^+(d)) \cup S_i^+(dc)$
14:        $S_i^- = (\cup_{d \in D_i} \mathsf{Neg}(d, S_i^+(d))) \cup \mathsf{Negp}(i, dc, S_i^+(dc), S_i^-(dc))$
15:        $c_i = W(rep_i(S_i^+), rep_i(S_i^-))$
16:    end while
17: end for
Output: the sequence $(c_1, \ldots, c_n)$ of classifiers
```

rithm 2.

Negative examples for dc should be carefully defined because dc is partially annotated. For instance, let us consider the case $i = 1$, that is the induction of the wrapper for the seed. In the first interaction: there are only two positive examples available for learning. Thus we have to guess for negative examples. The function Neg, used in the case of completely annotated documents, would give us all leaves in the current document which are not explicitly labeled as positive and the classifier learned would classify positive only leaves which are already known as positive!

To compute negative examples in the working document dc, we define the function Negp. It takes as input the index i, the working document dc, and the sets $S_i^+(dc)$ and $S_i^-(dc)$. When $i = 1$, it returns all leaves in $S_1^-(dc)$ together with all leaves whose path in the DOM tree is not the path of any leaf in $S_1^+(dc)$. When $i > 1$, two cases are to be distinguished: the first $i-1$ components may be factorized or not other several tuples of length i. In the first case, several tuples of length i are constructed over the same partial tuple of length $i-1$. The function Neg would generate false negative examples. Thus we need to guess which leaves may be good candidates to complete an $i-1$-ary tuple. Therefore, Negp returns all partial tuples of length i in $S_i^-(dc)$ together with all partial tuples of length i such that partial tuple of length $i-1$ is in S_{i-1}^+ and the path of the i^{th}

component is not the path of any i^{th} component in $S_i^+(dc)$. In the second case, Negp returns all partial tuples of length i in $S_i^-(dc)$ together with $\mathsf{Neg}(dc, S^+(dc))$.

Then the set of negative examples is computed in Line 14.

A classifier c_i is induced with the base supervised learning algorithm W updating the current partial wrapper $w_i = (c_1, \ldots, c_i)$. And the process iterates over calls to U and over i when U agrees with the current partial wrapper.

7 Experimental results

Evaluation Protocol An extracted tuple is correct if all its component values are correct. A component value is correct if it matches exactly the target component value. Performance of our wrapper induction system is evaluated using the average of precision (P), recall (R) and f-measure (F). They are computed with cross-validation techniques. In all experiments, the base learner for binary classifiers is Quinlan's C5.0 [15].

Experiments on RISE datasets Data sets in RISE correspond to the two first cases according to Section 2. Table 1 presents the experimental results where TD is the average number of tuples per document, and D is the number of documents used in the induction process. In this section and in the next section as well, at most two documents are given to the learner.

On BIGBOOK, OKRA and S20, Algorithm 2 obtain 100 of f-measure. For the problem S1 with the target relation (*company*, *price*, *product*), WIEN achieves 100 of f-measure while our system does not achieve 100 of f-measure because parts of textual values in leaves are needed for correct extraction. In the IAF dataset, with the target relation (*name*, *email*, *lastupdate*, *organization*, *altname*, *serviceprovider*), three of the six components of the target relation have missing values. Our system outperforms LIPX [17] (46 of f-measure) and WIEN which fails because it can't deal with missing values. On n-ary extraction tasks, we can not compare with systems like STALKER because it uses n unary wrappers and does not evaluate on tuples recombination. Further more it uses a post-processing, defined by the end-user, to construct tuples from component values. On unary extraction tasks, our system reaches the performances of STALKER (except on S1 as mentioned before). In particular on IAF, the f-measure of our system varies from 84 to 100, whereas it obtains 64 in the n-ary case. This show that tuples recombination is more difficult than components extraction.

Corpus	TD	D	P	R	F
BIGBOOK	18.29	1	100	100	100
OKRA	13,23	1	100	100	100
S20	32.7	1	100	100	100
S1	40.3	1	88.65	88.57	88.61
IAF	9	2	61.51	67.66	64.22

Table 1. Experimental results on RISE datasets.

Corpus	Case	TD	D	P	R	F
EXCITE 1	3,4	5	1	100	100	100
BBC	2,4	5	1	100	100	100
BLS 1	3	29	1	100	100	100
BLS 2	5	3.56	1	100	100	100
BLS 3	3	6	1	100	100	100
BLS 4	5	24	2	100	90	93.33
BEA 1	4,3	8.30	2	97.11	88.58	92.58
BEA 2	3	5.2	2	86.47	100	89.78
BTS	3	20	1	100	100	100

Table 2. Experiments on corporate Web sites.

Experiments on Web datasets We now consider datasets[6] obtained from several corporate Web sites. They were chosen to be representative of the tree organizations presented in Section 2. For the EXCITE 1 dataset, the relation to extract is (*town*, *day*, *weather*, *high*, *low*). There are five tuples, the component *town* is factorized while the others are stored in a rotated table. In BBC, the relation to extract is (*town*, *day*, *high*, *low*). The component *town* is factorized and the others are stored in a table. There are several datasets build from BLS with a target 3-ary relation. In BLS 1, BLS 2 and BLS 4, tuples are stored in a cross table, while in BLS 3 they are stored in a rotated table. In the BEA 1 benchmark (see Fig.1), the target relation is the 4-ary relation (*balance*,*year*,*country*,*export*). For each document of BEA 1, there are six tuples to extract. The component *country* is factorized among the six tuples. The other components are stored in a rotated table. In the BEA 2 benchmark, the target relation is the 3-ary relation (*year*,*balance*,*export*) stored in a rotated table. In the rotated table of BTS, the relation to extract is (*year*,*highway*,*oil*).

Experimental results are presented in Table 2. They show that Algorithm 2 succeeds with one or two completely annotated Web pages. Two pages were needed when there are large variations in the size of the tables and in the number of tuples to be extracted from one page to another.

Experiments in the interactive system We consider the interactive wrapper induction algorithm of Section 6 with its user simulator U. We compare the number of interactions in the completely annotated case and in the interactive framework. We denote by #F the number of interactions

[6]http://www.grappa.univ-lille3.fr/ marty/corpus.html

Corpus	TD	# I	# P	# F
EXCITE 1	5	13	25	25
BBC	5	7	13	15
BLS 1	29	25.8	32.40	87
BLS 2	3.56	7.60	12.20	10.68
BLS 3	6	13.60	21.60	18
BLS 4	24	16.40	30.20	288
BEA 1	8.30	32	56.40	66.40
BEA 2	5.2	9.4	15.20	31.2
BTS	8.30	29	45.00	60

Table 3. Interactive experiments

performed by the user with algorithm 2. It is defined to be $n \times m$ where m is the number of annotated tuples and n is the arity of tuples. In the interactive system, we denote by #I the number of component values selected or unselected by the user U, and we denote by #P the number of interactions performed by the user U. The selection of an example requires one interaction for the seed and two interactions for a partial tuple of length i (the tuple of length $i-1$ and the i^{th} component value). Thus a correction requires: 1 or 2 interactions for the seed; 2 or 4 interactions for the next steps.

We present in table 3 the number of interactions required in the two scenarios to achieve perfect wrappers on corporate Web sites. The number #I of examples selected or unselected by the user in the interactive framework is smaller than the number #F of interactions for a complete annotation. The number of interactions #P in the interactive framework is smaller than #F on datasets with many tuples per document. For benchmarks with few tuples to extract per document, like BLS 2, #P can be greater than #F because negative examples must be selected and because many partial tuples must be selected per tuple to be extracted.

We made experiments where the components order change, and we do not notice significant variations of the preformances.

8 Conclusion

We have presented a machine learning approach for n-ary relation extraction from semi-structured documents. Our algorithm combines the advantage that component extraction and tuples construction are made simultaneously in the wrapping procedure and in the induction process, without any post-processing.

The effectiveness of our approach is evident with completely annotated documents: we achieve state-of-the-art results on simple data organizations and also succeeds on intricate organizations like pivot table, rotated table and cross table where previous systems fail. Extraction and learning are fast. Furthermore, we have proposed an interactive framework in order to reduce the number of user actions and shown that our method is again improved: we are able to learn equally performant wrappers, with less effort for the user.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ICDM*, p. 337–348, 2003.

[2] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, p. 119–128, 2001.

[3] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. In *IJCAI Workshop on Grammatical Inference*, 2005.

[4] C. Chang and S. Lui. IEPAD: Information extraction based on pattern discovery. In *WWW*, 2001.

[5] W. Cohen, M. Hurst, and L. Jensen. *Web Document Analysis: Challenges and Opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific, 2003.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, p. 109–118, 2001.

[7] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, p. 577–583, 2000.

[8] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[9] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521 – 538, 1998.

[10] L. S. Jensen and W. Cohen. Grouping extracted fields. In *ATEM Workshop, IJCAI*, 2001.

[11] N. Kushmerick. *Wrapper Induction for Information Extraction.* PhD thesis, University of Washington, 1997.

[12] A. H. F. Laender, B. Ribeiro-Neto, A. S. Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, 2002.

[13] K. Lerman, C. A. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *ATEM Workshop, IJCAI*, 2001.

[14] I. Muslea, S. Minton, and C. Knoblock. Active learning with strong and weak views: a case study on wrapper induction. In *IJCAI 2003*, p. 415–420, 2003.

[15] R. Quinlan. Data mining tools see5 and c5.0, 2004. http://www.rulequest.com/see5-info.html.

[16] S. Raeymaekers, M. Bruynooghe, and J. Van den Bussche. Learning (k,l)-contextual tree languages for information extraction. In *ECML*, volume 3720 of *LNAI*, p. 305–316, 2005.

[17] B. Thomas. Bottom-up learning of logic programs for information extraction from hypertext documents. In Springer-Verlag, editor, *ECML/PKDD*, September 2003.

[18] I. S. I. University of Southern California. Rise, a repository of online information sources used in information extraction tasks, 1998. http://www.isi.edu/info-agents/RISE/index.html.

[19] Y. Zhai and B. Liu. Extracting web data using instance-based learning. In *WISE*, p. 318–331, 2005.

[20] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW*, p. 76–85, 2005.