XML Structure Mapping
Application to the PASCAL/INEX 2006 XML Document Mining Track[*]

Francis Maes, Ludovic Denoyer, and Patrick Gallinari

LIP6 - University of Paris 6
`firstname.lastname@lip6.fr`

Abstract. We address the problem of learning to map automatically flat and semi-structured documents onto a mediated target XML schema. We propose a machine learning approach where the mapping between input and target documents is learned from examples. Complex transformations can be learned using only pairs of input and corresponding target documents. From a machine learning point of view, the structure mapping task raises important complexity challenges. Hence we propose an original model which scales well to real world applications. We provide learning and inference procedures with low complexity. The model sequentially builds the target XML document by processing the input document node per node. We demonstrate the efficiency of our model on two structure mapping tasks. Up to our knowledge, there are no other model yet able to solve these tasks.

1 Introduction

Semantically rich data like textual or multimedia documents tend to be encoded using semi-structured formats. Content elements are organized according to some structure that reflects logical, syntactic or semantic relations between these elements. For instance, XML and, to a lesser extent, HTML allow us to identify elements in a document (like its title or links to other documents) and to describe relations between those elements (e.g. we can identify the author of a specific part of the text). Additional information such as metadata, annotations, etc., is often added to the content description leading to richer descriptions.

The question of heterogeneity is central for semi-structured data: documents often come in many different formats and from heterogeneous sources. Web data sources for example use a large variety of models and syntaxes. Although XML has emerged as a standard for encoding semi-structured sources, the syntax and semantic of XML documents following different DTDs or schemas will be different. For managing or accessing an XML collection built from several sources, a correspondence between the different document formats has to be established.

[*] This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

Note that in the case of XML collections, the schemas themselves may be known or unknown depending on the source. For HTML data, each site will develop its own presentation and rendering format. Thus even in the case of HTML where the syntax is homogeneous across documents, there is a large variety of formats. Extracting information from different HTML web sites also requires to specify some type of mapping between the specific Web sites formats and the predefined format required by an application.

Designing structure mappings, in order to define correspondences between the different schemas or formats of different sources is thus a key problem to develop applications exploiting and accessing semi-structured sources. This problem has been addressed for some times by the database and to a lesser extent by the document communities for different conversion tasks and settings. Anyway, the real world solution is to perform a manual correspondence between heterogeneous schemas or towards a mediated schema via structured document transformation languages, like XSLT. Given the multiplicity and the rapid growth of information sources, manually specifying the correspondence between sources is clearly a bottleneck to the process of document integration and reuse. Automating the design of these transformations has rapidly become a challenge.

This work was realized in the context of the PASCAL/INEX XML Document Mining Challenge[1]. This challenge proposes, as an extension to XML categorization and clustering, a track concerning the Structure mapping task. The goal of this task is to learn to transform HTML/flat document to an XML mediated schema as described previously.

We propose here to learn the transformation from examples. The learning system relies on a training set provided by the user. Each training example is made of an input document and the corresponding target document. The system will directly learn the transformation from these examples. Input documents may come with heterogeneous structures or simply with no structure at all. The manual specification of document mappings is replaced here with the development of a training set of transformed documents. This allows to consider problems where the input schema is not explicitly given or cases where this schema is too general so that no explicit mapping can be defined (this is the case for many HTML conversion applications). Besides, the proposed method only requires to provide a set of transformed documents and this task will be much easier than the manual development of a transformation script.

The structure mapping task we are solving is described in section 2. Our solution is detailed in part 3 and experiments performed on two different real world tasks are presented and discussed in section 4.

2 Structure Mapping Task

2.1 Description

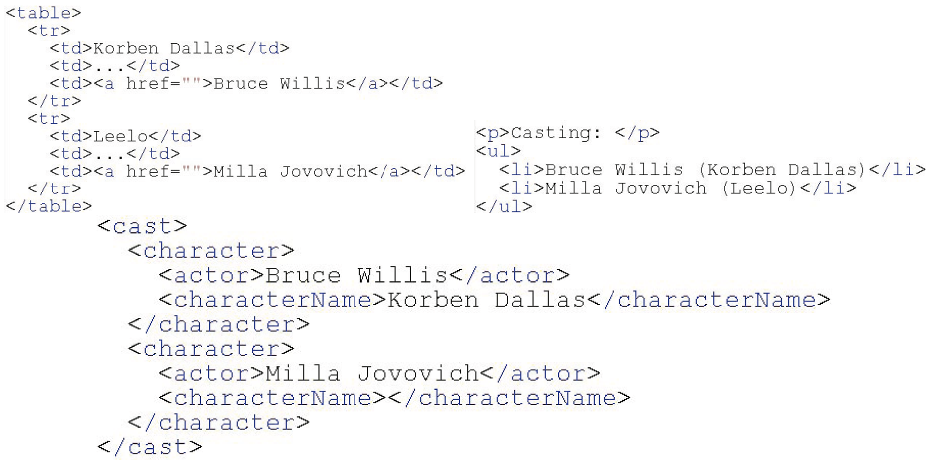
The structure mapping task addresses the problem of learning document transformations given a set of examples. The task is seen as a supervised learning

[1] http://xmlmining.lip6.fr

problem where inputs and outputs are semi-structured documents. Input documents may come from different sources and may take different formats like flat text, wikitext, HTML or different XML schemas. Output documents are expressed in XML. Given the set of learning examples, the aim is to learn an inference procedure able to convert any input document of the same family.

The structure mapping task encompasses a large variety of real applications like:

– *Semantic Web:* conversion from raw HTML to semantically enriched XML. Example sources include forums, blogs, wiki-based sites, domain specific sites (music, movies, houses, ...).
– *Wrapping of Web pages:* conversion from the relevant information in web-pages to XML.
– *Legacy Document conversion:* conversion from flat text, loosely structured text, or any other layout oriented format (*e.g.* PDF) to XML.



```
<table>
  <tr>
    <td>Korben Dallas</td>
    <td>...</td>
    <td><a href="">Bruce Willis</a></td>
  </tr>
  <tr>
    <td>Leelo</td>
    <td>...</td>
    <td><a href="">Milla Jovovich</a></td>
  </tr>
</table>
```

```
<p>Casting: </p>
<ul>
  <li>Bruce Willis (Korben Dallas)</li>
  <li>Milla Jovovich (Leelo)</li>
</ul>
```

```
<cast>
  <character>
    <actor>Bruce Willis</actor>
    <characterName>Korben Dallas</characterName>
  </character>
  <character>
    <actor>Milla Jovovich</actor>
    <characterName></characterName>
  </character>
</cast>
```

Fig. 1. Example of XML heterogeneity. The same movie description extracted from three sources: two HTML styles and one XML general movie schema.

Figure 1 illustrates an example of XML to XML conversion. As can be seen in this simple example the structure mapping task involves many different kind of elementary transformations, *e.g.* relabelling, node creation and suppression, node displacement. These actions can have global consistency constraints, *e.g.* conserving textual content order or being valid with respect to a DTD.

2.2 Formalization

Structure mapping consists in transforming $d_{in} \in D_{in}$ into an XML document $d^*_{out} \in D_{out}$ where D_{in} is the set of possible input documents and D_{out} is the set of possible output documents. For example, D_{in} is the set of all documents that are valid given a specific XML schema. As training data, an user provides a set of pair $\left\{(d^i_{in}, d^{i*}_{out})\right\}_{i \in [1,N]}$ where N is the number of examples, d_{in} is an input document in D^i_{in} and d^{i*}_{out} is the corresponding output document in D_{out}.

A structure mapping model is a function $f_\theta : D_{in} \rightarrow D_{out}$ that maps input documents into target documents. Such a model is parameterized by θ which is vector of real parameters. The quality of a structure mapping can be measured with a user supplied loss-function. This function is a dissimilarity measure between output documents : $\Delta : D_{out} \times D_{out} \rightarrow [0, 1]$. Good models will produce low loss. Learning is done by finding the parameters θ that minimize the empirical risk on the training set:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \Delta(f_\theta(d^i_{in}), d^{i*}_{out}) \tag{1}$$

Structure mapping models have to deal with two major difficulties. First, they have to support a large variety of transformations. The family f_θ must thus be expressive enough to express them. The other difficulty is related to the size of D_{out} which is exponential in the length of documents. A such problem makes full exploration of the output space intractable. The usual solution for this type of problem is to use dynamic programming techniques in order to efficiently explore the space of possible solutions. For the document transformation problem addressed here, the complexity of the inference step is so high that even dynamic programming does not lead to scalable solutions [1].

3 Proposed Model

Because of the exponential number of valid output documents that can correspond to a given input document, the simple strategy, which consists in generating all possible output to select the best one, is unrealistic. One way to break the complexity of the task, is to decompose it into simpler sub-problems. This can be achieved by the incremental structure mapping (ISM) algorithm we propose.

First, we describe the ISM process in the case of a simple HTML to XML structure mapping example. We then detail our general structure mapping inference algorithm. Finally, we present the learning algorithm that allows to find the parameters θ which minimize the empirical risk on the training set.

3.1 Incremental Structure Mapping

ISM rely on the idea that the structure mapping can be realized by considering successively each leaf of the input document and working out its position in the output document. The process is decomposed into successive elementary steps. Each of these steps do two things: reading/analysing a part of the input document and adding corresponding nodes in the current output document. Figure 2

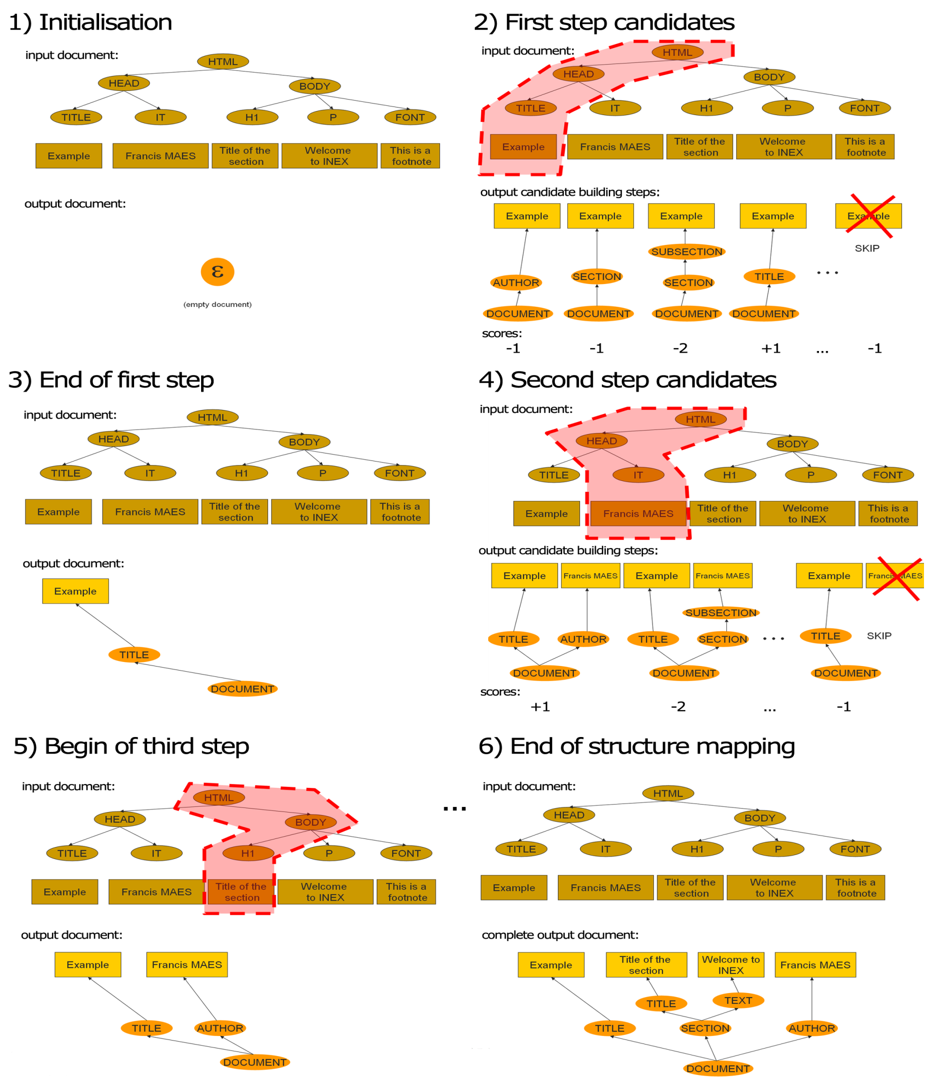


Fig. 2. 1)We initialize the process with the empty output tree (denoted ϵ). 2) First step: we focus on the first input leaf (with content "Example"). We enumerate building-step candidates. For each of this candidates there is an associated score. In this example the best building-step has score 1 and consists in adding "Example" as the output *Document Title*. 3) End of first step: the content "Example" has been added in the current output document. 4) Second step: we focus on the second input leaf, consider building-step candidates, where the best one is to add "Francis Maes" as the *Document Author*. 5) At the beginning of the third step, the *Author* has been added in the partial output tree. 6) Final state of the ISM process (after 5 steps) the current output tree is completed and can be returned to the user. We see in the final output tree that the *Section* nodes have been inserted between the two already existing nodes *Title* and *Author*.

Algorithm 1. Structure Mapping Inference

```
$d_{in} \in D_{in}$: the input document
$\phi : L_{in} \times B_{out} \rightarrow \Re^p$: the representation function
$\theta : \Re^p$: the learned parameters
 1: $d_{out} \leftarrow \epsilon$
 2: for all input leaf $n_i \in d_{in}$ do
 3:    candidates $\leftarrow$ computeOutputCandidatePaths($d_{in}$,$d_{out}$) $\bigcup$ SKIP
 4:    outputpath $\leftarrow$ argmax$_{c \in candidates} < \phi(n_i, c), \theta >$
 5:    if outputpath = SKIP then
 6:       continue (with next input leaf)
 7:    else
 8:       $d_{out} \leftarrow$ addNodeInOutputTree($d_{in}$, outputpath, $d_{out}$)
 9:    end if
10: end for
    return $d_{out}$
```

gives an illustrative example where an HTML document is being converted into XML.

We initialize the process with the empty output tree denoted ϵ. After initialization we iterate ISM steps, one for each input leaf. Thus an ISM step starts with selecting a new input leaf. Given this input leaf and the current output tree, we compute a set of building-step candidates. Each of these candidates is a particular way to insert the current input leaf content into the current partial output tree. It defines both new node labels and new node positions in the existing tree. Each building-step candidate has also an associated score. This score quantifies the immediate interest of executing a particular building-step. The model then chooses the best scoring building-step candidate, *execute* it and starts with the next ISM step.

The key idea with ISM is that instead of considering all valid output documents we consider only the valid building-steps at each step of the process. This way, the complexity of structure mapping only depends on the number of ISM steps and on the average number of candidates per step.

3.2 Inference Algorithm

The behavior of ISM is directly related to the scoring function of building-step candidates. This scores are only available for training document pairs. In the general case, these scores will be estimated using a linear predictor. Learning the ISM model from document pair examples is reduced to learning the ISM score predictor. Learning is described below, see 3.3. For the moment, let us consider that the model has been learned. Algorithm 1. shows the general ISM inference procedure.

The ISM inference algorithm has three parameters: the input document $d_{in} \in D_{in}$ and two parameters which defines the score predictor. In order to predict such a score, we first describe a (input leaf $\in L_{in}$, building-step candidate $\in$

B_{out}) pair using the ϕ function. This function produces a feature vector in $\Re^p$. Examples of such features are given in table 1. We produce features by combining state features with action features. To describe a (state,action) pair, we make the cross product between all state events and all action events. This way, ϕ produces sparse joint (state,action) representations where the number of distinct features p ranges usually from 10^3 to 10^6.

Table 1. Some examples of features which jointly describe a (input node, building-step) pair using the ϕ function. These features are usually binary (valued in $\{0, 1\}$) but general real valued features are also possible (*e.g.* the percentage of upper case words). The features are generated in a data-driven way: a feature is considered only once it is observed in the learning data. Depending on the corpora, the features vectors have from 10^3 to 10^6 distinct components.

Description	Value
We are processing the first input node and the building-step has labels *DOCUMENT TITLE*.	1
The input node has label *IT* and the building-step has labels *DOCUMENT AUTHOR*.	1
The current input leaf has 3 parents and the building-step inserts the node between a *TITLE* and a *SECTION*.	0
The last word of the current input node is “footnote” and the building-step is `SKIP`.	1
The last word of the input node is punctuation symbol and the building-step has labels *DOCUMENT SECTION TEXT*.	0
...	...

Given a description in $\Re^p$, the score is estimated by a dot product between the description vector and the parameters vector ($\theta \in \Re^p$). The dot product between two vectors is denoted $< ., . >$.

ISM inference is performed by iterating over input leaves. For each of this nodes, we first enumerate the set of building-step candidates (line 3). In order to include the possibility to skip some input leaves we also consider the `SKIP` building-step. `SKIP`ing a node means that this node will not be included in the output tree. In line 4, we estimate the scores of all candidates using our linear predictor. The best estimated building-step candidate is chosen. If the best building-step is `SKIP` we can continue with the next input leaf (line 6). In any other case, the building-step is *executed* with the *addNodeInOutputTree* function (line 8). This produces a new partial output tree d_{out}. Once all input leaves have been processed, the ISM is fulfilled and we return the current output tree d_{out} to the user.

3.3 Learning Algorithm

The ISM learning procedure is described in algorithm 2.. It aims at finding the parameters θ that leads to a good structure mapping inference procedure.

Algorithm 2. Structure Mapping Learning

```
$S = \left\{(d_{in}^{i}, d_{out}^{i*})\right\}_{i \in [1,N]}$: Training Set
$\phi : L_{in} \times B_{out} \rightarrow \Re^{p}$: the representation function
1: $\theta \leftarrow \mathbf{0}$
2: repeat
3:     $d_{in}, d_{out}^{*} \leftarrow$ pickTrainingPair($S$)
4:     $d_{out} \leftarrow$ structureMappingInference($d_{in}, \phi, \theta$)
5:     loss $\leftarrow \Delta(d_{out}, d_{out}^{*})$
6:     for all $n_i \in d_{in}$ do
7:         $\theta \leftarrow$ applyGradientCorrection($\theta$, $n_i$, loss)
8:     end for
9: until convergence of $\theta$
   return $\theta$
```

The algorithm has two parameters: the training set S of document pairs and the representation function ϕ described in previous section. The algorithm returns the learned parameters θ that can be used in inference.

ISM learning is done by iteratively evaluating and improving the parameter vector θ. In each iteration, we pick randomly a new training document pair (line 3). We then evaluate the current parameters θ by calling the inference procedure (line 4) and computing the resulting loss (line 5). We can now improve the parameters θ by applying a little correction for each ISM step that was performed during inference (line 6-8).

The details of the learning procedure are omitted here for the sake of clarity. However algorithms 1. and 2. relies on established machine learning techniques. Briefly, the ISM procedure can be modelled as a Markov Decision Process (MDP) [2]. MDPs provides a mathematical framework for modelling sequential decision-making problems. They are used in a variety of areas, including robotics, automated control, economics and in manufacturing. The fields of Reinforcement Learning [3] and Approximate Dynamic Programming [4] provides several learning algorithms for solving MDPs. Our learning procedure can be seen as a particular case of the Sarsa(0) algorithm. We differ the interested reader to [5] for more details.

4 Experiments

4.1 Tasks and Corpora

We present here experiments performed in the context of the INEX Structure Mapping Challenge. The challenge focuses on two corpora. The first is the INEX IEEE corpus which is composed of 12017 scientific articles in XML format. Each document comes from a journal (18 different journals). The documents are given in two versions: a flat segmented version and the XML version. The structure mapping task aims at recovering the XML structure using only the text segments as input. The segments are given in the exact order. The second corpora is made of more than 13000 movie descriptions available in three versions: two different XHTML versions and one mediated XML version. This corresponds to a scenario where two different websites have to be mapped onto a predefined mediated schema. The transformation includes node suppression and some node displacements.

In order to compare our model, we also made experiments on the Shakespeare corpora[2]. As in [6], we have randomly selected 60 Shakespearean scenes from the collection. These scenes have an average length of 85 leaf nodes and 20 internal nodes over 7 distinct tags. As a baseline, we implemented the model of [6] which is based on probabilistic context free grammars and maximum entropy classifiers. Due to its complexity (roughly cubic in the number of input leafs wheras ours is linear) this model cannot be applied to the others corpora.

Each corpus is split in two parts: 50% for training and 50% for testing. The table 2 summarizes the properties of our corpora.

Table 2. Description of the corpora used in our experiments. From left to right: the name of the corpus, the task, the number of documents, the mean number of internal nodes per document, the mean number of leaves per document, the number of disctinct tags.

Corpus	Tasks	Corpus size	Internal Nodes	Leaves	Labels
INEX IEEE	Flat $\rightarrow$ XML	12,017 docs	≈ 200	≈ 500	139
Movie 1	XHTML $\rightarrow$ XML	13,045 docs	≈ 78	≈ 31	16
Movie 2	XHTML $\rightarrow$ XML	13,038 docs	≈ 49	≈ 40	19
Shakespeare	Flat $\rightarrow$ XML	60 docs	≈ 20	≈ 85	7

4.2 Loss Function and Evaluation Measures

In order to evaluate the quality of structure mapping we have used two measures: $F_{content}$ and $F_{structure}$. The first measure reflects the quality of document leaves labelling. The second measure reflects the quality of the internal tree structure. Both measure are the mean of a F_1 score computed for all (predicted document, correct document) pairs. For $F_{content}$ we compute the F_1 score between leaves labels. This first measure is similar to the *Word Error Ratio* used in natural language. $F_{structure}$ is based on the F_1 score between all subtrees. This F_1 score between two trees is computed in the following way:

1. Build the set of all subtrees of the two trees. There is one sub-tree per node of the document
2. Compute recall and precision on the subtrees. Two subtrees are identical iff they have the same label, the same text (for leaves), and the same children trees (for internal nodes).
3. Compute the F1 score: $F1 = \frac{2*Recall*Precision}{Recall+Precision}$.

[2] http://metalab.unc.edu/bosak/xml/eg/shaks200.zip

This corresponds to a common measure in the natural language parsing field (under the name of *F1 parsing score*). Note that this measure decreases quickly with only a few errors. For example if there is only one labelling error in a leaf, the $F_{structure}$ measure typically equals to $\approx 80\%$.

4.3 Results

The loss-function $\Delta(d_{out}, d^*_{out})$ used for training the model is based on the $F_{structure}$ measure:

$$\Delta(d_{out}, d^*_{out}) = 1 - F_{structure}(d_{out}, d^*_{out})$$

Figure 4.3 shows the results obtained for the different experiments. All $F_{content}$ scores are greater than 75 % while the more difficult $F_{structure}$ is still greater than $\approx$ 60 %. These scores have to be contrasted with the intrinsic difficulty of the structure mapping tasks. For example for INEX, the only hints for predicted between more than hundred labels come from the textual content of the input document.

Corpus	Method	$F_{content}$	$F_{structure}$	Learning time	Testing time
INEX IEEE	ISM	75.8 %	67.5 %	$\approx$ 2 days	$\approx$ 2 s / doc
Movie 1	ISM	80.3 %	65.8 %	$\approx$ 17 min	$\approx$ 0.08 s / doc
Movie 2	ISM	75.4 %	57.0 %	$\approx$ 19 min	$\approx$ 0.07 s / doc
Shakespeare	ISM	89.4 %	84.4 %	$\approx$ 20 min	$\approx$ 0.02 s / doc
Shakespeare	PCFG+ME	98.7 %	97.9 %	$\approx$ 2 min	$\approx$ 1 min / doc

Fig. 3. Structure mapping results on the tree corpora. Two measure are used: $F_{content}$ and $F_{structure}$. Approximate learning and testing time are indicated - the experiments were performed on a standard 3.2Ghz Computer.

These results also show how fast ISM is at testing time. Most documents are processed in less than one second. This mean that ISM could be used with large scale corpora containing thousands or millions documents.

The Shakespeare database shows a comparison between ISM and our baseline called PCFG+ME (see part 4.1). ISM scores are less good than the baseline scores. A first explanation for this phenomenon is that PCFG+ME does a global optimization using Dynamic Programming. This has to be contrasted with ISM which performs a greedy search of the output tree. We also suspect ISM to suffer from over-fitting since there are few documents and many distinct features (see 1).

On the other side, ISM is approximatively thousand times faster in inference than the baseline. Moreover, due to its complexity the baseline cannot be applied to our real world corpora. We believe that in the context of heterogeneous information retrieval, fast inference time is much more important than perfect structure mapping.

5 Related Work

Several approaches to automating document transformation have been explored ranging from syntactic methods based on grammar transformations or tree transducers to statistical techniques. A majority of them only consider the structural document information and do not exploit content nodes. Even this structural information is used in a limited way and most methods exploit only a few structural relationships. Many of them heavily rely on task specific heuristics. Current approaches to document transformation are usually limited to one transformation task or to one type of data. Besides, most proposed techniques do not scale to large collections.

In the database community automatic or semi-automatic data integration — known as *schema matching* — has been a major concern for many years. A recent taxonomy and review of these approaches can be found in [7]. [8] describes one of the most complete approach which can handle ontologies, SQL and XML data. The matching task is formulated as a supervised multi-label classification problem. While many ideas of the database community can be helpful, their corpora are completely different from the textual corpora used in the IR community: all documents — even XML ones — keep an attribute-value structure like for relational database and are thus much smaller and more regular than for textual documents; textual data hardly appears in those corpora. With database corpora, finding the label of a piece of information is enough to build the corresponding tree because each element usually appears once in the tree structure. Document structure mapping, also shares similarities with the information extraction task, which aims at automatically extracting instances of specified classes and/or relations from raw text and more recently from HTML pages. Recent works in this field [9] have also highlighted the need to consider structure information and relations between extracted fields.

The structure mapping model proposed here is related to other Machine Learning models of the literature. Different authors ([10], [11]) have proposed to use natural language formalisms like probabilistic context free grammars (PCFG) to describe the internal structure of documents. Early experiments [1] showed that the complexity of tree building algorithms is so high that they cannot be used on large corpora like INEX. The work closest to ours is [6]. They address the HTML to XML document conversion problem. They make use of PCFGs for parsing text segment sequences and of a maximum entropy classifier for assigning tags to segments.

6 Conclusion

We have described a general model for mapping heterogeneous document representations onto a target structured format. This model learns the transformation from examples of input and target document pairs. It is based on a new formulation of the structure mapping problem based on Deterministic Markov Decision Processes. This formulation allows us to deal with a large variety of tasks ranging from the automatic construction of a target structure from flat documents to the mapping of XML collections onto a target schema. The model operates fast and scales well with large collections. We have shown its efficiency on two real world large scale tasks.

References

1. Denoyer, L., Wisniewski, G., Gallinari, P.: Document structure matching for heterogeneous corpora. In: SIGIR 2004. Workshop, Sheffield (2004)
2. Howard, R.A.: Dynamic Programming and Markov Processes. Technology Press-Wiley, Cambridge, Massachusetts (1960)
3. Sutton, R., Barto, A.: Reinforcement learning: an introduction. MIT Press, Cambridge (1998)
4. Si, J., Barto, A.G., P.W.B., II, D.W. : Handbook of Learning and Approximate Dynamic Programming. Wiley&Sons, INC., Publications, New York (2004)
5. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems, vol. 8, pp. 1038–1044. The MIT Press, Cambridge, MA (1996)
6. Chidlovskii, B., Fuselier, J.: A probabilistic learning method for xml annotation of documents. In: IJCAI, pp. 1016–1021 (2005)
7. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: A brief survey. AI Magazine, Special Issue on Semantic Integration (2005)
8. Doan, A., Domingos, P., Halevy, A.: Learning to match the schemas of data sources: A multistrategy approach. Maching Learning 50(3), 279–301 (2003)
9. Califf, M.E., Mooney, R.J.: Bottom-up relational learning of pattern matching rules for information extraction. J. Mach. Learn. Res. 4, 177–210 (2003)
10. Young-Lai, M., Tompa, F.W.: Stochastic grammatical inference of text database structure. Mach. Learn. 40(2), 111–137 (2000)
11. Chidlovskii, B., Fuselier, J.: Supervised learning for the legacy document conversion. In: DocEng '04, pp. 220–228. ACM Press, New York (2004)