Reverse Engineering for Web Data: From Visual to Semantic Structures

Christina Yip Chung
Verity Inc.
Sunnyvale, CA 94089, USA
cchung@verity.com

Michael Gertz
Department of Computer Science
University of California, Davis
Davis, CA 95616, USA
gertz@cs.ucdavis.edu

Neel Sundaresan
NehaNet Corp.
San Jose, CA 95131, USA
neel@nehanet.com

Abstract

Despite the advancement of XML, the majority of documents on the Web is still marked up with HTML for visual rendering purposes only, thus building a huge amount of "legacy" data. In order to facilitate querying Web based data in a way more efficient and effective than just keyword based retrieval, enriching such Web documents with both structure and semantics is necessary.

This paper describes a novel approach to the integration of topic specific HTML documents into a repository of XML documents. In particular, we describe how topic specific HTML documents are transformed into XML documents. The proposed document transformation and semantic element tagging process utilizes document restructuring rules and minimum information about the topic in form of concepts. For the resulting XML documents, a majority schema is derived that describes common structures among the documents in the form of a DTD. We explore and discuss different techniques and rules for document conversion and majority schema discovery. We finally demonstrate the feasibility and effectiveness of our approach by applying it to a set of résumé HTML documents gathered by a Web crawler.

1 Introduction

The vision of the Semantic Web of the future is that it will be the medium for the exchange, sharing, and retrieval of information in a meaningful and effective way [8]. However, before this becomes reality, we have to deal with large volumes of "legacy" data marked up in HTML. Though HTML is used as "the" language for markup, even documents that conceptually follow a common schema are marked up for visual rendering purposes only, and in different ways due to diverse authorship and goals of the people writing these documents. This makes it quite difficult to automate the processing of HTML documents in terms of data retrieval and integration.

XML, on the other hand, tries to alleviate these problems by providing several means to enrich documents (or data in general) by more structural and semantic information. Several sophisticated approaches to processing XML data have been developed in the past few years, in particular XML query languages and query processing techniques [1, 5, 6, 10, 25]. These approaches, however, assume that the data is already presented in either XML or related semistructured data formats [2], and do not appropriately take the existence of the large amount of data only marked up in HTML into account.

Recently, several approaches have been developed to close this gap, that is, to translate HTML data into XML data using *wrappers*, e.g., [3, 4, 18, 24]. The approaches differ in what input is required to generate a wrapper for a specific source, whether the wrapper can handle heterogeneous sources, and in particular in the approach to translate HTML documents into XML documents. However, in most of these approaches it is difficult to easily incorporate and exploit properties of tree structures (as which HTML documents can be considered). Furthermore, most approaches do not take the semantics of HTML objects into account by utilizing conceptual information such as topic specific concepts and keywords.

In order to process XML documents efficiently, e.g., for querying or integration, it is vital to know the schema underlying the documents. Such a schema is often not explicit, e.g., in the form of a DTD, but inherent in the document structure. Recently, several approaches to the discovery of schemas have been proposed in [17, 22, 23, 26, 27]. The approaches differ in the type of schema discovered and assumptions made about the semantics of structures in the underlying XML documents.

In this paper, we propose a novel approach that effectively combines the two steps of data conversion (wrapping) and schema discovery. The advantage of tightly coupling these steps is that information that has been gathered during the conversion can be utilized appropriately in the schema discovery process. In our approach, we focus not on arbitrary collections of HTML documents but topic specific documents that pertain to a particular domain of interest and have been gathered by a topic specific Web crawler. Thus we anticipate a highly heterogeneous (in terms of visual representations) collection of HTML documents. As our "wrapping" approach, we apply several heuristics to recover XML documents. For this, we do not only rely on visual representations (markup), as done by related approaches, but make use of the HTML tree structure as well as some topic specific information such as topic concepts and keywords.

Another important contribution of the proposed approach is the schema discovery method that determines a so-called *majority schema*. Well-known approaches to schema discovery and specification for XML data typically either determine an upper bound schema in the form a Data Guide [19], comprising all structures that can be found in the XML documents, or a lower bound schema [2], comprising structures that can be found in all documents. Only a very few approaches have been proposed that discover schemas between the two extremes [23, 27]. The majority schema we derive from a set of XML documents describes structures that occur frequently in the documents. Thus, there may be a few input XML documents that do not conform to the schema. We apply different types of heuristics to discover and unify such structures. We also show how a DTD including information about element ordering and repetitive structures can be derived from a majority schema.

Compared to a Data Guide or lower bound schema, there are several advantages of utilizing a majority schema. For example, if the input XML documents need to be integrated into some kind of XML repository, the majority schema can be used to translate the input XML documents so that they conform exactly to the majority schema. A Data Guide or lower bound schema obviously would not suffice in this case. Even for query optimization and index structures on XML documents, a majority schema can provide the right level of detail necessary to realize these tasks efficiently, rather than to provide either too much detail or not enough details necessary. In particular, for the integration of differently structured XML documents, a more flexible approach to schema discovery as a guidance for the integration is needed. In this paper, we present such an approach.

Related Work. Our approach is most closely related to the aforementioned approaches to wrapping HTML documents and the discovery of schema structures from XML data. Several types of data source and document wrappers have been suggested in the literature. W4F [24] and YAT [9] are instances of manual wrappers which require users to specify exactly how to extract data from HTML documents through some wrapper language. The problems of manual wrappers are: (1) they assume the data is in a known format or variants of the same format. They are applicable for data from the same data source, but not for data from a large collection of diverse sources (or the Web in general). (2) The format of the data may change over time. Every change of format would require a new handcrafted wrapper. Unlike these manual wrappers, our document conversion process does not rely on a specific format of the documents.

In approaches related to automatic wrapper generation, e.g., [3, 16], users give examples of the data extraction process and the system learns the extraction rules. They assume that records in an HTML document or all HTML documents follow the same format. This is inapplicable for data from numerous diverse sites. In our document conversion approach, we only assume that records *within* a document follow some regular patterns. This is justified because usually there is only one author for an HTML document. Different documents, however, may follow different formats.

As indicated above, there have also been several approaches to the discovery of schemas from semistructured data and XML documents. The approach described in [27] tries to discover an approximate version of a graph schema as in [23] by clustering similar objects together. A similar notion of clustering related objects into a class is considered in [22]. Nevertheless, these approaches take the data-centric view of [23] as a model of semistructured data. Hence they ignore the inherent tree structure of HTML and XML documents which, very often, plays a significant role in the semantics of these documents. The approach described in [26] is probably the closest to our approach. [26] views an HTML document as a tree and decomposes it into a sequence of label paths. The structure of the document is captured as a sequence of frequently occurring label paths. However, [26] tries to model the tree structure too precisely. In doing so, prevalent regular patterns may be missed since they are buried under too many details. In addition, there is a penalty in computational complexity.

Structure of the Paper. In Section 2, we present our approach to extracting XML documents from topic specific HTML documents. Section 3 discusses the derivation of a majority schema and DTD from such XML documents. Section 4 presents results we obtained by applying our approach to topic specific HTML documents. Section 5 contains final remarks and a discussion of future work.

2 Extraction of Schematic and Semantic Structures

The first task in integrating topic specific, heterogeneously structured HTML documents is to transform them into XML documents. This is accomplished by the document conversion process. The XML documents then embed not only logical schematic structures but also carry semantic information in form of element names. In Section 2.1, we give a short overview of the approach, which is based on topic specific concepts and diverse domain-independent document restructuring rules. In Section 2.2, we introduce the notion of topic specific concepts as minimal user input to our approach. Section 2.3 discusses the restructuring rules that map components of HTML documents to proper (nested) structures and elements of XML documents.

2.1 Overview

Although the input HTML documents are heterogeneous in terms of how topic specific information is represented through HTML markups (due to different authorship), the documents exhibit certain domain-independent properties. An HTML document basically consists of two types of elements: *block level* elements and *text level* elements. The former describe the document structure in terms of headings, lists, text containers, tables etc. whereas text level elements mark up text inside block level elements, e.g., based on font markup tags. Together, these elements specify information carrying objects of an HTML document at different levels of abstraction through object nesting. The structure of a document thus can be viewed as follows: objects of a higher level of abstraction are described by objects of finer levels of abstraction. Typically, the meaning of an object is not directly associated with the object itself based on meaningful tags, but is buried in the text (information content) as well as in associated structures (block level context structures).

There are two major tasks in the document conversion process. First, one has to identify objects at the right level of abstraction. Second, identified objects and their information content need to be translated into XML elements which carry the original information content and in particular more meaningful tags. The transformation of an HTML document into an XML document is executed by applying the above steps to the HTML document. For the first task, the identification of objects in the document can occur in two ways: (1) through identifying certain keywords in text components, and/or (2) taking block level elements into account that only have the purpose of visually rendering the object. Keyword-like information is provided by domain knowledge pertinent to the topic the HTML documents are about. An object identified in this way is then mapped to an element and structure of the corresponding XML document by applying several domain-independent rules.

2.2 Topic Concepts

A basic assumption underlying our approach is that the collection of HTML documents pertains to a particular, relatively narrow (in terms of concepts) topic. Although the collection is heterogeneous in terms of how documents are structured, the information content of the documents is quite homogeneous. Thus, we assume that there is some domain knowledge specific to the topic the documents are about. The two types of domain knowledge we consider are *topic concepts* and *concept constraints*. While the former is mandatory user input to the transformation process, the latter is not required but can be used to optimize the document conversion process.

We assume a set Con of *topic specific concepts*. Such concepts provide the domain for tags to be chosen as elements in XML documents. In particular, they reflect the meaning of objects that typically constitute a topic specific document at different levels of abstraction. Concepts are provided by a single user initiating the document transformation process.

With each concept $c \in Con$, a set of *concept instances* is associated, which also includes the name of the concept. Concept instances specify text patterns and keywords as they might occur in topic specific HTML documents. Assume, for example, a collection of r*é*sum*é*s as HTML documents. One obviously expects some information about institutions where the person received his or her degree(s). Thus `institution` would be a concept and it would also be used as element name in resulting XML documents if information about institutions can be identified based on concept instances. For this concept, typical instances would be *University, College* etc. Similar types of concept instances can be given for concepts such as `degree`, `programming-skills`, `date` etc.

In a document, different objects can be associated with the same concept. This typically holds for topic independent concept such as `date` or any other measurement-type concept. However, the context of the concepts then differs, that is, they represent homonyms. Homonyms can play different roles in different contexts. For example, in order to detail information about the concept `education`, `date` can be used to chronologically organize this information, whereas for other concepts, `date` does not exhibit such a property. This as an important aspect in authoring documents and is taken into account in the document restructuring process.

Some concept instances are often already present in order for the topic specific crawler to gather respective documents from the Web. Thus, such instances can be reused for the transformation process. The aggregation of concept instances into concepts and the specification of further concept instances can easily be done by the user after inspecting a few of the retrieved HTML documents. As we will show in Section 4, already a relatively small number of concepts and instances leads to accurate XML documents obtained by the document conversion process.

In order to further guide and optimize the document restructuring process, a user can specify *concept constraints*. Such constraints are optional to the process. They basically describe how concepts as information carrying objects can be structured. In our approach, we employ three simple types of constraints. Let $c_1, c_2 \in Con$. $parent(c_1, c_2)$, $sibling(c_1, c_2)$, and $depth(c_1)\ \Theta\ d$ ($\Theta \in \{=, <, >\}$) specify that c_1 is a (not necessarily direct) parent of c_2, c_1 and c_2 are sibling concepts (i.e., at the same level of abstraction), and concept c_1 can only occur at a certain depth in the XML document, respectively. All predicates can also be negated, thus allowing to specify atypical properties of concepts. Note that since concept constraints are optional, they do not have to be complete. A complete specification of all admissible and prohibited relationships between concepts would basically resemble a common logical layout among the XML documents to obtain. This would be much too restrictive since the goal of our approach is to provide a flexible, easy to employ framework for translating heterogeneous HTML documents into XML documents.

2.3 Document Tree Restructuring

We now turn to the core document transformation task in which concepts are employed to restructure HTML documents into XML documents. For this, we consider an input HTML document as XML document. In particular, we assume that the document (1) is represented as an ordered tree, and (2) each HTML and XML element has an attribute named `val` of type `CDATA`. The translation of an HTML document into an XML tree can occur in a straightforward manner by adopting the Document Object Model [15].

A core characteristic of HTML documents is that HTML markups only serve visual rendering purposes. That is, they do not describe the logical, semantic layout of documents. The visual clues given by such markups, however, can be used to recover the logical document structure. That is where the novelty of our proposed document restructuring approach lies. Restructuring rules are applied to the input tree and restructure the initial HTML document into an XML document in a top-down/bottom-up fashion. The rules basically restructure the tree components by replacing HTML block level and text level elements with XML elements carrying concept names as tags.

We distinguish between two types of restructuring rules: *text rules* and *structure rules*. The former deal with identifying concepts in text as it occurs in the input HTML document. The latter deal with recovering contextual properties of objects that have been related to concepts but are still "embedded" in HTML block level elements. In particular, the structure rules ensure that the resulting document only contains XML elements that carry concept names as element names.

2.3.1 Text Rules

We employ two text rules, the *tokenization rule* and the *concept instance rule*. The rules are applied consecutively to the document in a top-down fashion. The purpose of the tokenization rule is to take a text node (also called topic sentence) in the HTML tree and to decompose the text into tokens. Formally, a tokenization rule takes an HTML text node and replaces it by $n \geq 1$ *token nodes* of the pattern `<TOKEN>text</TOKEN>`. The rationale behind this rule is that text describing some high level concept typically consists of information components related to lower level concepts. Often such components are separated by punctuation delimiters, such as semicolon ";", comma "," etc. Consider, for instance, the topic sentence "University of California at Davis, B.S.(Computer Science), June 1996, GPA 3.8/4.0" representing an information object about the educational background in a résumé. According to the text rule, this topic sentence is tokenized into token nodes and further processed by the concept instance rule. The number and order of tokens obtained from a text depends on the number and choices of delimiters considered in tokenization.

For each token obtained through the tokenization rule, the *concept instance rule* checks whether the token can be related to a concept. There are several tools that have been developed in the area of Information Retrieval and that are suitable for this task. In our current implementation, concept instances are identified from tokens in two ways: (1) by synonyms, and (2) by a multinomial Bayes classifier [12].

For (1), it is simply checked whether for a concept instance a match (synonym) can be found in the token. For Bayes classifier, the user gives examples on how to associate tokens with concept instances by labeling some input HTML documents. Based on these examples, the Bayes classifier computes the statistics of associating words in the token with concept instances. Given a new résumé document, the classifier classifies each token as a concept instance with the highest probability using the statistics computed from the training documents.

For a token identified by the tokenization rule, there are two cases to be considered.

1. A concept instance has been identified in the token node, i.e., the instance has been associated with a concept $C \in Con$. Then the token node `<TOKEN>text</TOKEN>` is replaced by an XML element of the pattern `<C val="`*text*`"/>`. That is, the concept name C is chosen as the element name and the token text becomes the value of the element attribute `val`. The text associated with token nodes is always assigned to this attribute. This, however, is only an implementation related issue and can easily be devised differently.

Applying the concept instance rule to the tokenization of the above topic sentence leads to four sibling XML elements in the document, assuming that instances for the concepts `institution`, `degree`, `date`, and `GPA` have been defined appropriately:

```
<INSTITUTION val="University of Cali-
                  fornia at Davis"/>
```

```
<DEGREE val="B.S.(Computer Science)"/>
<DATE val="June 2001"/>
<GPA val="GPA 3.9/4.0"/>
```

If more than one concept instance is found in a token, the token is further decomposed. Such a scenario can occur if delimiters are not used consistently in topic sentences or certain delimiters are not considered by the tokenization rule. Assume, for example, a token node of the pattern `<TOKEN>`*text*$_1$ *text*$_2$ *text*$_3$ *text*$_4$ *text*$_5$`</TOKEN>` such that no delimiter occurs in or between two *text*$_i$ and *text*$_j$. Furthermore, assume that *text*$_2$ and *text*$_4$ have been associated with concept C_1 and C_2, respectively. In this case, the text consisting of an identified concept instance up to the next identified instance is associated with a concept. For the rightmost identified instance, the remaining text is taken. The text before the first identified instance ($text_1$ above) is passed as value of attribute `val` to the node's parent node. In the above case, we thus replace the token node by the two XML elements <C_1 `val=`"*text*$_2$ *text*$_3$"`/>` and <C_2 `val=`"*text*$_4$ *text*$_5$"`/>`. In such scenarios, concept constraints describing typical sibling relationships can be employed in order to determine a proper decomposition.

2. *No concept instance can be found in the token node, i.e, the token cannot be associated with any concept.* In this case, the token node is deleted and the *text* value is passed to the parent node as value for the attribute `val`. The justification for this is that child nodes detail information represented by parent nodes at a lower level of abstraction. The same principle has been applied in case 1 above where $text_1$ has been passed to the parent node. In that respect, the concept instance rule ensures that no text information is lost but kept in a proper context during the document restructuring process. It should be noted that one can also pass context information (e.g., depths in the tree or sibling nodes) about the unidentified token to the parent node in order to better segment (and eventually query) multiple unidentified tokens.

Concept instances are identified *after* tokenization. The reason is that since tokenization partially identifies objects describing a higher level object, applying concept instance identification to tokens can improve the accuracy of identifying information the instances describe. The less instances are identified, the more values for the attribute `val` are passed to parent nodes. It is thus advisable to use the ratio between identified and unidentifiable tokens (or tokens that have been classified as "unknown" in case of the Bayes classifier) as a feedback to the user who then in turn has to provide more training data to the classifier or associates more concept instances with concepts.

While the previous two rules are concerned with the text data embedded in HTML objects, the following rules try to recover structural properties from an HTML document and to restructure the document so that it eventually only contains properly nested XML element structures.

2.3.2 Structure Rules

We employ two types of structure rules. These rules basically consider the HTML block level element context of already identified XML elements and use this information to restructure the document such that only logical (XML) structures remain. That is, all HTML elements will be replaced by XML elements. Both rules below operate on the tree representation of a document in a consecutive fashion.

Grouping Rule. In an HTML document, certain block level elements, called *group tags*, give hints to the grouping of semantically related objects. For example, `block` elements (such as `p, hr`), `list` elements (such as `dl,ul`), `heading` elements (such as `h1,h2, ...`) and `table` elements (such as `tr,td`) are often used to divide the data into sections (or information objects) of the same level of abstraction. It is thus reasonable to preserve a group of objects occurring at the same level of abstraction in an HTML document in the resulting XML document. The following restructuring rule accomplishes this.

Given sibling nodes $N_1, \ldots, N_k$ in the document tree that all have the same markup tag. Then all sibling nodes $S_1, \ldots, S_n$ that occur between N_i and N_{i+1} are grouped under a new node with the (temporary) label `GROUP`, and this node becomes a child of node N_i. All sibling nodes right to N_k are grouped in the same way. Note that the nodes $S_1, \ldots, S_n$ must not necessarily have the same tag. Groups of semantically related information objects thus "sink" in the tree structure and are put into a logical nesting. In the current implementation, we assign weights to certain block level elements. For example, grouping right siblings of nodes marked with `h1` has a higher priority than grouping right sibling of nodes marked with `p` at the same level. Since each group sinks down and the rule operates in a top-down fashion, groups related to `p` nodes then will be considered at the next lower level.

Consolidation Rule. The consolidation rule is the final rule that is applied to the document tree. The rule operates bottom-up on the document tree, eliminating any remaining HTML markups (such as `font` tags) as well as nodes that have been introduced temporarily, e.g., by the grouping rule. Before applying this rule, leaf nodes in the document tree either already represent XML elements (obtained through the concept instance rule) or HTML markups that do not carry any meaningful information for the document. The rule is based on the following observation. Consider a group of objects that all reside at the same level in the ordered document tree. Often the first object in such a group of semantically related objects describes the concept of this group.

Assume a node N that is marked up with an HTML tag. If N does not have any children, it is deleted from the tree. Otherwise, there are two options which depend on the properties of the node's children, say $S_1, \ldots, S_k$. For instance, if an HTML node tag N is known to exhibit a list structure (such as the `list` elements `ul`, `dl` or the `table` element), its children are likely to be objects of the same level of abstraction. A more trivial case is when the children already carry the same XML element name. In both cases, the sibling relationship between $S_1, \ldots, S_k$ is maintained by "pushing up" all the S_is, replacing node N. Thus the nodes S_i become children of N's parent. If $S_1, \ldots, S_k$ do not exhibit a similar structure, then N is replaced by its *first* child node that has been related to a concept. Figure 1 illustrates this aspect (most values for the attributes `val` have been left out for simplicity). The upper part shows a document before applying the consolidation rule.

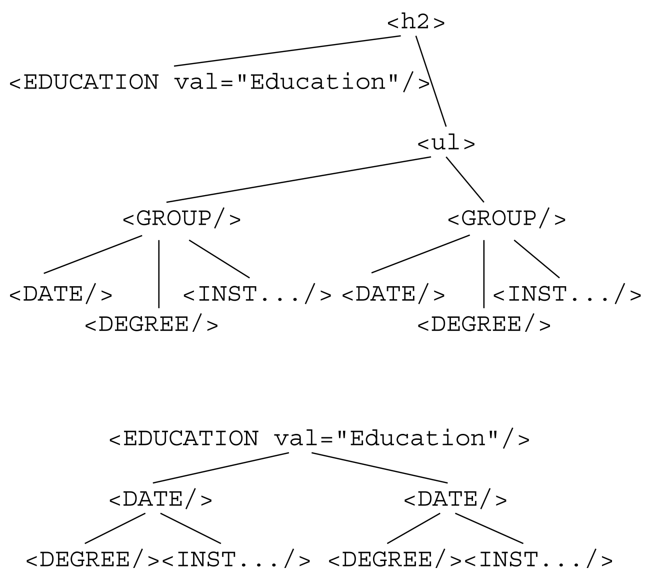

Figure 1. Replacing Non-concept Nodes by Concept Nodes

The consolidation rule starts bottom-up with the left-most `GROUP` node. Since the children of the nodes labeled `GROUP` do not exhibit similar structures, each such node is replaced by its first child, i.e., the nodes labeled with `DATE`. Next, under the node marked `ul`, all children now exhibit the same structure since they are all labeled with `DATE` and `ul` is known to exhibit a list structure. Thus, their sibling relationship is maintained by removing the node `ul` and pushing them up in the tree, i.e., they now become children of the node labeled `h2`. Finally, the rule is applied to node `h2`. In this case, the node's children do not exhibit similar structures. Thus, the first child node (`EDUCATION`) replaces this node so that we end up with the subtree shown in the lower part of Figure 1.

The consolidation rule thus exactly describes how non-concept nodes are replaced by concept child nodes and how sibling relationships are maintained while such replacements occur. For this, the rule can also utilize existing concept constraints in order to determine whether a node (concept) can become a parent or sibling of another node.

2.4 Resilience of Restructuring

The above text and structure rules utilize some heuristics which are based on the semantics of HTML tags and experiences obtained by applying the rules to different topic specific collections of HTML documents. Although the heuristics are resilient to a certain extend in case input HTML documents are not well-formed (e.g., nesting of heading elements), experiments show that applying HTML cleansing tools (such as HTML Tidy) can improve the accuracy of resulting XML documents. Some of the rules can be refined even further. For example, the concept instance rule and cases where more than one concept instance can be found in a token node can be improved by applying certain natural language language processing techniques. Furthermore, certain rules can be devised differently based on the specific type of HTML block level element. The presented rules are of a very general type. Their characteristics to employ numerous hints on visual document representations for deriving logical document structures is very powerful and applicable to many document collections that exhibit in particular deeply nested document structures, one major advantage of our approach over existing wrapper approaches.

3 Schema Discovery

Although the XML documents obtained by the process described in the previous section are marked up with the same XML elements, the documents still may exhibit heterogeneous structures due to different authorship. The next step is to determine common schema structures, called *majority schema*, among a set of XML documents. After an overview of the basic idea underlying the proposed schema discovery approach, in Section 3.2, we describe the computation of so-called *frequent paths* from input XML documents. Frequent paths describe a majority schema for the documents. In Section 3.3, we outline how a DTD is derived from a majority schema.

3.1 Overview

For the discovery of common structures among a set of XML documents, we use an approach similar to [26] in which ordered trees representing XML documents are mapped to paths. Unlike [26], where paths are built based on node identifiers, we choose to adopt some simplifications: (1) paths are sequences of node labels instead of node identifiers, (2) an ordered tree is reduced to a *set* of paths emanating from the tree's root. That is, initially we do not consider ordering and repetitive information of sibling nodes. The reason to adopt these simplifications is motivated not only by efficiency concerns, but also by the fact that the input documents do not all have exactly the same schematic structure. For the discovery of common structures, it is thus desirable to introduce a certain degree of flexibility at the beginning so that regular schematic patterns (frequent paths) evolve. After prevalent patterns are discovered, heuristics will be employed to recover further structural information such as ordering and repetition in order to determine a common schema in the form of a DTD.

3.2 Frequent Paths

We assume that the schematic structure of an XML document D is represented by an ordered tree $T = (V, E, root)$. We also assume a function $label$ that maps nodes in V to element types E. Node labels E are concept names used in obtaining the XML documents from HTML documents. A *node path* $v = v_i, v_{i+1}, \ldots, v_j,\ v_k \in V$ describes a sequences of nodes in T, not necessarily starting from $root$. The *label path* of a node path p is determined by the labels of its nodes, i.e., $labelpath(v) := label(v_i) \circ label(v_{i+1}) \circ \cdots \circ label(v_j)$. Thus two different node paths can have the same label path. In order for the proposed schema discovery method not to be too biased towards multiple occurrences of the same path in only a very few documents, we represent an XML document T as a set of paths, denoted $paths(T)$, emanating from $root$.

We will, however, need information about the multiplicity of sibling nodes of a given node in a path. This information will later be used to determine repetitive elements for a DTD. Let $p = p_1 \circ p_2 \circ \ldots \circ p_n$ be a path in $paths(T)$. For a path prefix $p = p_1 \circ \ldots \circ p_k, k \leq n$, $\langle p, num \rangle$ denotes the number of sibling nodes of type p_k for node p_{k-1}. Recording the multiplicity of child nodes does not cause any computational overhead since all paths in a given tree need to be considered anyway.

The search space $\mathcal{S}$ to be considered for the discovery of common schematic structures among a set $\mathcal{D}_{\mathsf{XML}}$ of XML documents is determined by the multiset of label paths that can be derived from the trees associated with the documents. Let T_{D} denote the tree associated with a document D. Then $\mathcal{S} := \{\{p \mid p \in paths(T_{\mathsf{D}}) \ \wedge\ \mathsf{D} \in \mathcal{D}_{\mathsf{XML}}\}\}$.

We use the occurrence frequency as a measure to evaluate how common a label path is among all documents (see also Figure 2). Instead of considering only the occurrences of the paths in $\mathcal{S}$, we consider the occurrences of all their prefixes up to the complete paths because several trees may share the same path prefix but not the complete path. For a label path p, the function $support$ is defined as follows. Let $freq(p, \mathcal{S})$ denote the number of occurrences of the prefix p in all paths in $\mathcal{S}$.

$$support(p) := \frac{freq(p, \mathcal{S})}{\mid \mathcal{D}_{\mathsf{XML}} \mid} \in \mathbb{R}[0, 1]$$

It seems reasonable to use the support as the main criterion to determine whether or not a path prefix is common, as other approaches to approximate schema discovery inherently assume. Since the support of a path prefix naturally decreases with its length, in particular "close" to leaf nodes, a more practical approach utilizes a *support ratio* for a path prefix, which is defined as $supportRatio(p) := support(p)/support(p')$, where $p = p' \circ p'', p'' \in E$. We assume that for a path p consisting only of the *root* node, $supportRatio(p) := 1$. The discovery of common schematic structures can now be guided by specifying two thresholds $supThreshold, ratioThreshold \in \mathbb{R}[0, 1]$ a label path p has to satisfy in order to be frequent. For a label path p to be a member of the set of *frequent paths*, denoted by $\mathcal{F}$, p has to satisfy the conditions that $support(p) \geq supThreshold$ and $supportRatio(p) \geq ratioThreshold$.

Obviously, the higher $supThreshold$, the more selective and thus common are the schema structures discovered. In particular, once a path (prefix) does not satisfy $supThreshold$, all its superpaths need not be considered. Note that if $support(p) = 1$, then p can be found in every document, and if $support(p) > 0$, then there exists at least one document that contains that path.

Figure 2 shows the trees A, B, and C. Figure 3 shows the set of corresponding label paths $\mathcal{S}$ (as a tree structure).

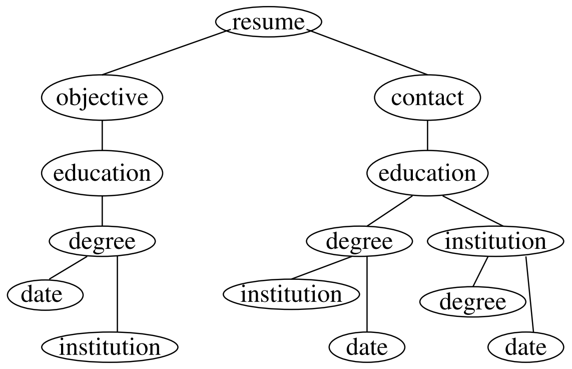

Figure 3. Tree representation of a set of labels paths

It should be noted that similarly structured components in a schema discovered by this approach can be further unified. Because of space limitations, this optional step is not described in this paper but can be found in [13].

3.3 Obtaining a DTD

The set of frequent paths discovered constitute a *majority schema* for the XML documents. As common for tree or

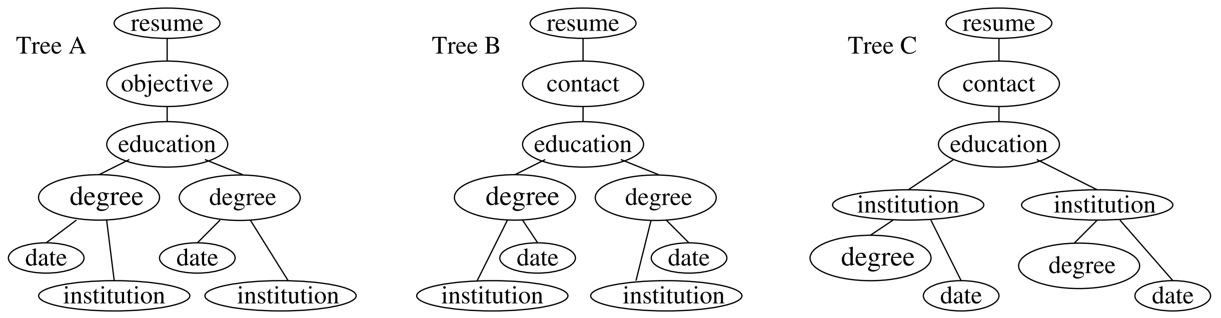

Figure 2. Trees Representing XML Documents and Set of Label Paths

graph-based schemas, however, the schema does not contain any information about repetitive elements or ordering. In this section, we outline how this information can be obtained from frequent paths and be encoded in a DTD. Recall that since the schema discovered is a majority schema, there can be documents that do not conform to that DTD.

While a set $\mathcal{F}$ of frequent paths can easily be mapped to a tree structure $T_{\mathcal{F}}$, in order to describe this structure in the form of a DTD, we have to specify a content model for each type of node in that tree. The content model, besides nesting information, also allows the specification of ordering and repetition information, a distinguishing feature between graph schemas and DTDs. More precisely, for a node e' in $T_{\mathcal{F}}$, we would like to associate a content model α with e', where α is an expression of the form $\alpha := e \mid \alpha_1 \mid \alpha_2 \mid \alpha_1, \alpha_2 \mid \alpha_1? \mid \alpha^* \mid \alpha^+$.

e is an element from E, the α_is are content models. We use two rules to determine the content model for elements.

Ordering Rule. Given a set of label paths $\{pq_1, \ldots, pq_m\}$ from $\mathcal{F}$ that share the same maximum prefix p. The q_is are the different child nodes of p as they occur in $T_{\mathcal{F}}$. Let $\mathcal{D}^p_{\mathsf{XML}}$ be all XML documents that contain the prefix p. Then the ordering of the child elements $q_1, \ldots, q_m$ for p is determined by the average position an element q_i occurs as child of p in the documents $\mathcal{D}^p_{\mathsf{XML}}$.

Algorithmically, the ordering rule first is applied to the child nodes of the root in $T_{\mathcal{F}}$ and then consecutively to each inner node. The result for a node e in the DTD is an element content specification of the form `<!ELEMENT e (qi1, qi2, ...., qim)>` with the `qi`s being the child nodes of e in $T_{\mathcal{F}}$. This specification is only temporary since it is refined by the rule below. An efficient computation of an ordering can be supported by an appropriate index structure on the input XML documents. That is, for each path and node, the index contains pointers to the positions in XML documents that contain that node. Such an index structure can easily be built while the set $paths$ is computed for each XML document (see Section 3.2).

Repetitive Elements. Because each path in $T_{\mathcal{F}}$ is a frequent path, no element should be optional. This viewpoint can reduce the question to whether an element typically occurs only once or multiple times as a child of element e in the input XML documents. In order to answer this question, we utilize the multiplicity information we recorded for each node in XML documents (see Section 3.2). It should be noted that the same multiplicity information can be used to introduce optional elements, if this is desired in a specific application scenario.

Let $p = p' \circ e$ be a prefix of some frequent path in $T_{\mathcal{F}}$ and let e' be the last element in the label path p'. We assume that the placement of e in the content model for e' has already been determined. For each document D that contains the path p and for each of p's children, we first decide whether it is a repetitive structure or not. That is,

$$rep(T_{\mathsf{D}}, p) := \begin{cases} 1, & \text{if for prefix } p \text{ in } path(T_{\mathsf{D}}) \text{ with} \\ & \langle p, num \rangle,\ num \geq repThreshold \\ 0, & \text{otherwise} \end{cases}$$

Obviously, the value for $repThreshold$ has to be greater than 1 for e to be repetitive. Empirical studies prove the value 3 to be useful, a fact that also has been observed in [17]. The multiplicity of e in the content model for e' then can be determined as follows. $mult(e) :=$

$$\frac{\mid \{\{rep(T_{\mathsf{D}}, p) \mid rep(T_{\mathsf{D}}, p) = 1 \wedge \mathsf{D} \in \mathcal{D}^p_{\mathsf{XML}}\}\} \mid}{\mid \mathcal{D}^p_{\mathsf{XML}} \mid}$$

If for a path p and element e, $mult(e) \in \mathbb{R}[0, 1]$ is greater than a specified threshold, say 0.5, e is specified in $e's$ content model as `e+`, otherwise, it is just `e`. In the above description we do not consider repetitive structures of more general types, e.g., of the form `(e1,e2)*`. The discovery of such patterns has been discussed in detail in [17]. We recently included similar computations into our approach.

4 Evaluation

In this section, we briefly describe an empirical study to evaluate the effectiveness, scalability, and feasibility of our approach. For this, we used resumés marked up in HTML and which have been gathered by a Web crawler [20]. This crawler was programmed to crawl the Web looking for HTML documents that looked like resumés. We ran our experiments on a Pentium 266MHz processor with 196MB main memory and 512KB cache. There are 24 concept names and a total of 233 concept instances specified as domain knowledge. The following annotation of tags is used to extract schematic structures from the documents: punctuation in tokenization {',',':',';'}, group tags {`h1,h2,h3,h4,h5,h6,h4,div,p,tr,dt,dd,li title,u,strong,b,em,i`}, and list tags {`body,table,dl,ul,ol,dir,menu`}.

4.1 Data Extraction Accuracy

We evaluated the accuracy of our document restructuring rules by counting the number of wrong parent-child and sibling relationships in the extracted tree. We reorder the nodes in the extracted tree in order to convert it to the correct tree. In doing so, we may move a node and its siblings together to make up for one parent-child relationship that has been incorrectly identified. This is counted as *one logical error*. 50 resumé documents are manually inspected to count the number of errors in the extracted trees (Figure 4). The average number of errors in each hypertext document is 3.9. The average number of concept nodes (XML elements) in a document is 53.7. The average percentage of error nodes in a hypertext document with respect to the total number of concept nodes is 9.2%. In other words, our restructuring rules have an accuracy of 90.8%.

4.2 Concept Constraints

We specified two classes of concept constraints to test our approach. First, we assume that a concept name cannot appear more than once along any label path. Second, we divided the set of concept names into two sets – title names and content names. Title names are likely to be the title of a resumé, and hence can only occur as first level nodes in the tree. Content names describe the content of title names and hence can only occur at depth d with $d > 1$ in the tree. Out of the 24 concept names, 11 are title names and 13 are content names. We also specified that no concept can occur at a depth greater than 4.

Without any relationships and constraints specified, exhaustive enumeration and testing of all possible label paths up to length 4 against the input HTML documents would explore $24^5 - 1 = 7962623$ nodes. With the above simple constraints specified, the search space is dramatically reduced to $1 + 11 + 11 \times 13 + 11 \times 13 \times 12 = 1871$ nodes, which corresponds to 0.023% of the exhaustive search space. Without extending nodes with zero support, the actual number of nodes explored is 73, representing a mere 0.0009% of the search space. The quality of the common schematic structures discovered is not compromised by reduction of the search space explored. On the contrary, it is of higher quality since constraints help to filter out noise. This serves to demonstrate the effectiveness and usefulness of concepts and constraints.

4.3 Scalability

We measured the scalability of our approach against the size of the HTML documents, the number of nodes in the documents and the number of concept nodes. We tested our schema discovery approach for datasets of increasing number of resumé documents chosen at random from the repository of resumés our crawler gathered. The size of the dataset is up to 380 resumé documents, corresponding to 699 nodes and 187 concept nodes. Results are shown in Figure 5, which shows that the running time bears a very strong linear relationship with the number of concept nodes, as expected. The running time also scales linearly with the number of nodes and the number of documents.

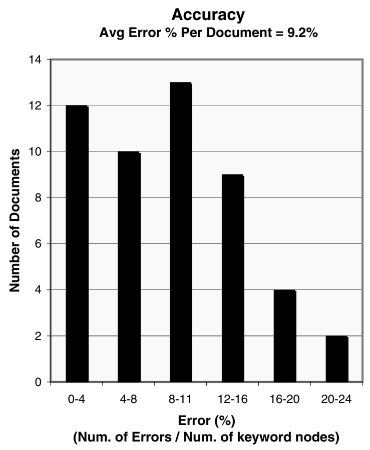

Figure 4. Accuracy of Heuristics

4.4 Sample Run

To demonstrate the feasibility of our approach, we attempt to discover the schema for over 1400 resumé documents marked up in HTML. A fragment of the DTD discovered (having a total of 20 element) is shown below. Manual inspection of the DTD reveals that the schema discovered indeed agrees with common sense of how a schema for resumé documents should look like.

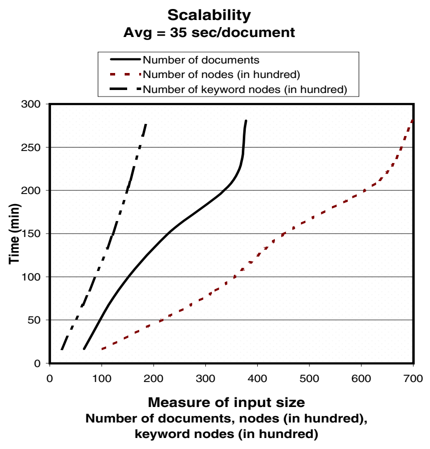

Figure 5. Scalability

```
<!ELEMENT resume  ((#PCDATA), contact+,
            objective, education+, courses,
            experience+, awards, skills,
            activities+, reference)>
<!ELEMENT contact     (#PCDATA)>
<!ELEMENT objective   (#PCDATA)>
<!ELEMENT education   ((#PCDATA),
              institute, date-entry))>
<!ELEMENT institute   (#PCDATA)>
<!ELEMENT date-entry  ((#PCDATA), degree)>
<!ELEMENT degree      (#PCDATA)>
<!ELEMENT courses     ((#PCDATA), date+)>
<!ELEMENT date        (#PCDATA)>
```

5 Conclusions and Future Work

In this paper, we presented a novel framework for the integration of topic specific but heterogeneously structured HTML documents. For the document extraction and transformation process, we employ diverse types of document restructuring rules that utilize structural and visual properties of HTML markups in order to derive properly nested XML documents from input HTML documents. The minimal user input to this process are topic specific concepts and concept instances. The novelty of our document restructuring approach lies in using information about how HTML markups are typically used to render information in documents, and how such markups can be related to logical document structures. Such structures eventually constitute XML documents.

We have shown how from the obtained set of XML documents a new type of schema, called majority schema, can be efficiently discovered. Such an approximate schema describes structures which are common among the input XML document. We have also shown how a DTD can be derived from a majority schema. We outlined feasibility results obtained so far for the approach by applying the document transformation and schema discovery methods to a relatively large collection of topic specific HTML documents.

A first working prototype of the proposed framework including a Document Mapping Component, which has not been described in this paper, has been implemented and is described in [11]. The Document Mapping component, which is described in detail in [13], converts non-conforming XML documents using a tree-edit distance algorithm so that they eventually conform to the derived DTD and can easily be integrated into an XML document repository. Our results show that such conversions are only reasonable by using a majority schema; Data Guides or lower bound schemas do not suffice for this task. Regarding the proposed document restructuring rules, we are currently investigating more sophisticated heuristics and automated discovery methods for concepts and concept instances from HTML documents. In particular, we are developing different methods to automatically extract concept instances from a training set of HTML documents and thus to further automate the process.

Finally, we are investigating the application of all three steps (1) HTML to XML document transformation, (2) schema discovery, and (3) schema guided XML document mapping to more complex and in particular broader types of topics. We are in particular interested in incorporating linkage structures among HTML documents. We hope that this will give our approach the flexibility to integrate even more heterogeneous, multi-topic HTML documents into XML repositories on a large scale basis. Naturally, the goal of this more recent investigation is definitely not to transform the whole Web into a repository of XML documents, but to build XML repositories capturing linked HTML documents pertaining to broader topics such as product catalogs or University Web sites.

References

[1] S. Abiteboul. Querying semi-structured data. In *6th Int. Conf. on Database Theory (ICDT '97)*, LNCS 1186, 1–18, Springer-Verlag, 1997.

[2] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web – From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[3] B. Adelberg. NoDoSE: A tool for semi-automatically extracting semi-structured data from text documents. In *Proc. ACM SIGMOD*, 283-294, 1998.

[4] N. Ashish, C.A. Knoblock. Wrapper generation for semistructured Internet sources. In *Proc. Workshop on Management of Semistructured Data*, 1997.

[5] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April, 1997.

[6] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79, March 2000.

[7] P. Buneman, S.B. Davidson, M.F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *6th Int. Conf. on Database Theory (ICDT '97)*, LNCS 1186, 336–350, Springer, 1997.

[8] T. Berners-Lee, M. Fischetti, and M. Dertouzos. *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, October 1999.

[9] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *Proc. ACM SIGMOD*, 177–188, 1998.

[10] D. Chamberlin, D. Florescu, and J. Robie. Quilt: An XML query language for heterogeneous data sources. In *3rd Int. Workshop on the Web and Databases*, LNCS 1997, 2000.

[11] C. Chung, M. Gertz, and N. Sundaresan. Quixote: Building XML Repositories from Topic Specific Web Documents. In *Fourth Int. Workshop on the Web and Databases (WebDB'2001)*, 2001.

[12] S. Chakrabarti. Data mining for hypertext: A tutorial survey. In *SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM*, 1, 2000.

[13] C. Chung. Data Extraction and Integration of Semistructured Documents. PhD Thesis, University of California, Davis, California, 2001.

[14] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. Computer Networks 31(11-16): 1155-1169, 1999.

[15] Document Object Model (DOM) Level 2 Core Specification, W3C Recommendation, November 2000.

[16] H. Davulcu, G. Yang, M. Kifer, and I. Ramakrishnan. Computational Aspects of Resilient Data Extraction from Semistructured Sources. In *19TH ACM Symp. on Principles of Database Systems*, 136–144, 2000.

[17] M.N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. Xtract: A system for extracting document type descriptors from XML documents. In *Proc. ACM SIGMOD*, 165–176, 2000.

[18] J. Gruser, L. Raschid, M.E. Vidal, and L. Bright. Wrapper generation for web accessible data sources. In *3rd IFCIS Int. Conf. on Cooperative Information Systems (CoopIS)*, 14–23, 1998.

[19] R. Goldman and J. Widom. Data guides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, 436–445, 1997.

[20] IBM Almaden Research Center. IBM: All searches start at Grand Central. *Network World*, Nov. 1997.

[21] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. In *Workshop on Management of Semistructured Data*, 1997.

[22] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proc. ACM SIGMOD*, 295–306, 1998.

[23] S.. Nestorov, J.D. Ullman, J.L. Wiener, and S.S. Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE'97)*, IEEE, 1997.

[24] A. Sahuguet and F. Azavant. Looking at the Web through XML glasses. In *Proc. of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS)*, 148–159, 1999.

[25] V. Vianu. A Web Odyssey: From Codd to XML. In *Proc. of the 20th Symposium on Principles of Database Systems (PODS 01)*, 1–15, 2001.

[26] K. Wang and H. Liu. Discovering association of structure from semistructured objects. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):353–371, 2000.

[27] Q. Wang, J. Yu, and K. Wong. Approximate graph schema extraction for semi-structured data. In *6th Int. Conf. on Extending Database Technology (EDBT 2000)*, LNCS 1777, Springer, 2000.