TreeScope: Finding Structural Anomalies In Semi-Structured Data

Shanshan Ying  
Advanced Digital Sciences Center  
shanshan.y@adsc.com.sg

Barna Saha  
University of Massachusetts Amherst  
barna@cs.umass.edu

Flip Korn  
Google Research  
flip@google.com

Divesh Srivastava  
AT&T Labs–Research  
divesh@research.att.com

ABSTRACT

Semi-structured data are prevalent on the web, with formats such as XML and JSON soaring in popularity due to their generality, flexibility and easy customization. However, these same features make semi-structured data prone to a range of data quality errors, from errors in content to errors in structure. While the former has been well studied, not much attention has been paid to structural errors, which can impact applications quite severely.

In this demonstration, we present TREESCOPE, which analyzes semi-structured data sets with the goal of automatically identifying structural anomalies from the data. Our techniques learn robust structural models that have high support, to identify potential errors in the structure. Identified structural anomalies are then concisely summarized to provide plausible explanations of the potential errors. The goal of this demonstration is to enable an interactive exploration of the process of identifying and summarizing structural anomalies in semi-structured data sets.

1. INTRODUCTION

Semi-structured data are prevalent on the web and in NoSQL document databases, with formats such as XML (eXtensible Markup Language) and JSON (JavaScript Object Notation) soaring in popularity due to their generality, flexibility and easy customization. However, these benefits come at the cost of being prone to a range of data quality errors, from errors in content to errors in structure. Errors in content have been well studied in the literature \cite{1,3}, while very little attention has been paid to errors in structure, with most of it focusing on well-formedness and validity \cite{6}. This is based on the assumption (used in well-structured data) that once data are valid according to the specified schema (DTD or XSD for XML data, JSON Schema for JSON data), there can be no structural errors. We have found this assumption to often be incorrect.

In our work we observe that DTD/XSD specifications for heterogeneous XML data sets tend to be quite liberal, allowing semantically incorrect (though syntactically valid) data to creep into the data sets. The existence of such errors can lead to incorrect results on queries \cite{9}, and can impact applications that rely on semantically correct data quite severely. We present illustrative examples of such errors in the well-known and widely-used DBLP data set\footnote{http://dblp.uni-trier.de/}.

Example 1. The tree rooted at dblp in Figure 1 represents a fragment of the DBLP data set. Each non-leaf tree node under dblp corresponds to an element and each leaf node corresponds to a text value in the dataset. Six publication instances are shown: three inproceedings conference papers, a journal article, a www publication, and a conference proceedings.

All nodes in red are examples of semantically incorrect but syntactically valid data. For example, the first inproceedings is a conference paper (by Maleki and Mohades), and has a spurious author element (which is meaningful for journal papers, but not for conference papers); the third inproceedings conference paper (by Kasi) has two crossref elements with the same text value, one of which is redundant; the journal article (by Johansson and Johansson) and the www publication (by Tschira) use editor tags incorrectly instead of author tags.

DBLP has a DTD which requires each publication to be one of eight types (article, book, etc.), enumerates the valid contained elements to be one of author, editor, and so on, but imposes no additional restrictions, making it a very liberal DTD. As a result, many structural errors exist in DBLP despite the data being valid according to the DTD \cite{7}.

Liberal schemas tend to be specified for two reasons. First, specifying precise schemas that can identify all kinds of errors in structure is a difficult task, even for a domain as well understood as DBLP bibliographic data; second, attempting to specify precise schemas is likely to make the schema overly complex and potentially conservative, making it more likely to reject semantically correct data as invalid, which is quite undesirable as well.

In this work, we present TREESCOPE, which incorporates novel techniques to analyze semi-structured data sets with the goal of automatically identifying potential structural errors in the data. A key insight is that it is not necessary to learn the precise schema to identify structural errors. Rather, it is sufficient to learn robust structural models of subsets of the semi-structured data with high support, and identify structural anomalies as violations of the learned models. A structural model $M$ is a triple $(c,t,f)$, where $c$ is a context path expression, $t$ is a target tag, and $f$ is the expected frequency (e.g., OneOrMore, Zero, AtMostOne) of the number of occurrences of target tag $t$ (e.g., editor, number) in each of the elements $e$ in the result $E$, of evaluating the context path expression $c$ (e.g., `/dblp/inproceedings`).
TREESCOPE learns robust structural models through a controlled exploration of the lattice structure of context path expressions that have high support, computing frequency distributions of candidate target tags, and finding those structural models that exhibit a significant skew in their frequency distributions. Structural anomalies are then identified as elements satisfying the context path expression of the learned robust structural model, whose frequency of a target tag is an outlier (i.e., has few occurrences in the skewed frequency distribution) compared to the expected frequency.

**Example 2.** Consider a model with a simple path as its context: \( M_1 = \langle /\text{dblp/author}, \text{number}, \text{Zero} \rangle \). The model \( M_1 \) says that each inproceedings should have 0 occurrence of the number element. In Figure 2, the second and the third inproceedings satisfy \( M_1 \), while the first inproceedings (by Maleki and Mohades) violates it. Such an element can be identified as a structural anomaly with respect to \( M_1 \).

Context path expressions can be more complex than simple paths, taking advantage of predicates and wild cards, as illustrated by the expression \( /\text{dblp/article[not (cdrom)]} \) in Figure 3.

Since the number of structural anomalies in real semi-structured data (such as DBLP) can easily be in the thousands (if not more), enumerating the structural anomalies one by one is not necessarily the best way to present the results. Hence, TREESCOPE uses summarization techniques based on greedy weighted set cover heuristics over lattice structures \(^2\) to concisely summarize the structural anomalies for presentation to domain experts, who can then distinguish true structural errors from structurally rare, but semantically correct data.

In our demonstration, VLDB attendees will be presented with the robust structural models and structural anomalies learned by TREESCOPE from the widely-used DBLP and Mondial data set. Attendees will be able to interactively explore the process of identifying, explaining and summarizing structural anomalies in these data sets, using the TREESCOPE tool.

### 2. TREESCOPE SYSTEM ARCHITECTURE

The TREESCOPE system integrates a visualization frontend with algorithmic backend components: the system architecture is shown in Figure 2 and we discuss each component in more detail below.

#### 2.1 Backend Components

There are two components in the backend, one for anomaly detection and the other for anomaly summarization. We first introduce the **Anomaly Detection** component, which systematically explores candidate structural models and identifies structural anomalies, and then describe the **Anomaly Summarization** component.

---

\(^2\) [http://www.dbis.informatik.uni-goettingen.de/Mondial/](http://www.dbis.informatik.uni-goettingen.de/Mondial/)
Horizontal Expansion: add a directed “horizontal” edge from \( v \) to \( v' \) such that the context path expression \( c' \) of \( v' \) is obtained from \( c \) by (i) specializing the wild card in \( \text{child::* in a nodetest of } c \) by an element name \( \text{elemName} \), or (ii) conjointing an atomic predicate of the form \( \text{child::* (if } c \text{ does not contain a wild card), child:* or not (child::elemName) to } c \).

Horizontal Shrinkage: add a directed “horizontal” edge from \( v' \) to \( v \) such that the context path expression \( c' \) of \( v' \) is obtained from \( c \) by (i) generalizing the element name in \( \text{child::elemName in a nodetest of } c \) by the wild card (*), if \( c \) does not contain a wild card, or (ii) removing an atomic predicate from \( c \).

Vertical Expansion: add a directed “vertical” edge from \( v \) to \( v' \) such that the context path expression \( c' \) of \( v' \) is obtained from \( c \) by appending a new step \( \text{child::* to } c \), if \( c \) does not contain a wild card.

Note that each component of the subgraph induced by only the horizontal edges (i.e., horizontal expansion and horizontal shrinkage) is a lattice. Note that if there is a directed edge from \( v \) to \( v' \) in a lattice, the set of elements \( E_{v'} \) in the result of evaluating the context path expression \( c' \) of \( v' \) is a subset of the set of elements \( E_c \) in the result of evaluating \( v \)’s context path expression \( c \).

Pruning the Search Space
The above search graph generation process terminates when all distinct context path expressions (in the restricted XPath language) have been generated. As an optimization, TREESCOPE stops expanding the structural model when either it is consistent or it has insufficient support (\(|E_c| \leq \theta \)), as a consistent model will never lead to any anomalies and a model with low support is not robust.

Structural Anomaly Identification
The structural anomalies for different target tags can obviously be different. For this reason, TREESCOPE identifies structural anomalies for each target tag \( t \), and for each lattice in \( G_t \). Given a lattice, vertices that have no outgoing horizontal edges are referred to as leaves. TREESCOPE identifies structural anomalies based only on the structural models corresponding to the leaves of the lattices, since these are the most specific robust structural models learned.

2.1.2 Structural Anomaly Summarization
Each robust structural model can give rise to a set of structural anomalies. However enumerating all structural anomalies is not necessarily the best way to present results, as they could be in the thousands. Hence, TREESCOPE summarizes the structural anomalies before presenting them to an expert. The summarization is done separately for each target tag \( t \), and for each lattice in \( G_t \). The summarization problem is modeled as a weighted set cover problem over lattice structures, to cover all the structural anomalies using a small number of structural models, possibly corresponding to non-leaf vertices. This can be solved using a greedy weighted set cover heuristic 5.

2.2 Frontend Visualization
The frontend visualization is designed to help users understand and interactively explore the lattices. Users will first be presented with a list of structural anomalies, as shown in Figure 3. Each row in the list consists of: 1) context path expression \( c \) and tag \( t \) of the structural model; 2) expected frequency \( f \); this serves as the explanation for the structural anomalies; 3) the number of structural anomalies w.r.t the structural model. For example, the first row indicates 388 structural anomalies for \( \text{inproceedings} \) with \( \text{child pages} \), with zero \( \text{author} \) elements, as the expected frequency of authors is OneOrMore.

Note that such leaves may have outgoing vertical edges.

For the DBLP2013 dataset, we set \( \theta = 10,000 \) and \( \alpha = 0.1\% \), and for the Mondial 2009 dataset, we set \( \theta = 150 \) and \( \alpha = 5\% \).
We mark in total 7230 and 151 anomalies as true errors in DBLP and Mondial respectively. From Table 1, we can see that, in both datasets, TREESCOPE detects more true anomalies, and has a higher recall. But TREESCOPE loses a little in precision on DBLP dataset, due to detecting more anomalies on cdrom and note, which are marked as rare occurrences, but not true errors.

Users can play with different threshold settings using slider bars to get a better result. When the frequency threshold $\theta$ and skew threshold $\alpha$ vary, the list of anomalies detected changes. A larger $\theta$ reduces the number of vertices in the lattice, while a larger $\alpha$ may result in a larger subset of elements identified as anomalies. If the user wants to see more potential anomalies, she may lower the $\theta$ value and increase the $\alpha$ value. In contrast, if many anomalies reported are actually false positives, the user should try a smaller $\theta$ to filter out less frequent elements.

### 3.2 Demo Walkthrough

The TREESCOPE system is deployed on our server. In the demonstration, users may explore the structural anomalies from both DBLP and Mondial datasets, using the Chrome browser.

Users may click the blue “Distribution” icon to see the frequency distribution, to find whether it is skewed, and in which way. There are three frequency groups, zero (Group 0), one (Group 1), and more than one (Group $2^+$) in the chart, with y-axis as the number of elements in each group. This would help the user to verify whether the expected frequency in the model makes sense. In this example, the user will find that only 363 elements are from Group 1. By comparing this number with the large number of (more than 1 million) elements that fall in Group 0, she may believe that the 363 elements are worthy of subsequent verification. To understand the anomalies, samples from different groups will be retrieved by clicking the green “Information” icon. At most 10 elements will be fetched in each group. In this example, the user will see some papers published in ICCS 2010 in Group 1, and will judge whether these conference papers should have the number element.

Recall that we apply a greedy algorithm for summarization. To find out how well the summarization algorithm works, for each lattice, the user could use the highlight button (below the vertical zoom in/out bar) to show the coverage of each lattice vertex. All vertices picked by the summarization algorithm are in black. By clicking on any of the vertices, all reachable vertices from the clicked one will be connected with edges. For instance, Figure 4 shows a lattice with number as the target tag, 4 vertices in red circles are picked by the summarization algorithm. Once a vertex is clicked, we will see its original color (blue), meaning that its expected frequency is zero. By following the edges we can trace how many blue vertices in the leaf layer are covered. Thus, users will see clearly the coverage of each vertex, and see if there are any interesting vertices missing from the summarization.

TREESCOPE also permits predicate and wild card in the context path expression. The user can compare the differences in generated lattices and detected anomalies by switching on/off the predicate and wild card functions from our visualization tool.

### 4. RELATED WORK

Recently several approaches have been proposed for schema inference on XML documents. XTract [4] generates a set of candidate regular expressions from each element. The most concise one is selected as the best answer. [8] uses multiple approaches to generate probabilistic string automata representing regular expressions, by application of inductive inference theory. Geert et al [2] propose to infer a concise DTD from the XML data. But all these works assume the training data is not only correct but also fairly complete, which is not realistic in our XML data sets. As a consequence, these approaches are not practically viable for automatically identifying structural errors in semi-structured data, as we plan to show in the TREESCOPE system.

### 5. CONCLUSIONS

In this demonstration, we present TREESCOPE, which analyzes semi-structured data sets with the goal of automatically identifying structural anomalies from the data, by learning robust structural models through a controlled exploration of the lattice structure. An interesting interactive online visualization tool is designed to help users explore the process of identifying and summarizing structural anomalies. Anomalies from real datasets, such as DBLP and Mondial, are available online for users to play with.

An interesting direction of future work is to identify meaningful repairs of the identified structural anomalies.

### 6. REFERENCES


