# TreeScope: Finding Structural Anomalies In Semi-Structured Data

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

Flip Korn
Google Research
flip@google.com

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

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 very 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, little attention has been paid to structural errors.

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 [1, 3], while very little attention has been paid to errors in structure, with most of it focusing on well-formedness and validity [6]. This is based on the assumption 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 errors in their structure. 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 [9], and even worse result in poor data-driven decisions.


*Proceedings of the VLDB Endowment,* Vol. 8, No. 12


We present illustrative examples of such errors in the well-known and widely-used DBLP Computer Science bibliography data set.

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

*All nodes in red are examples of semantically incorrect but syntactically correct data. For example, the first* `inproceedings` *is a conference paper (by Maleki and Mohades), and has a spurious* `number` *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 be one of the 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 being valid according to the DTD [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 possibly 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 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 $\langle c, t, f \rangle$, 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 $E_c$ (e.g., a subset of `inproceedingss`) 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

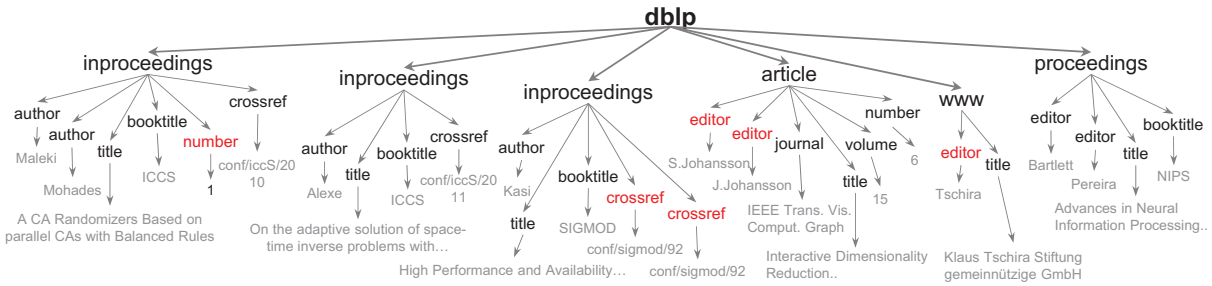


**Figure 1:** Examples from DBLP

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. Context path expressions can be more complex than simple path expressions, taking advantage of predicates and wild cards.

EXAMPLE 2. *Consider the model with a simple path as its context:* $M_1 = \langle$`/dblp/inproceedings`, `number`, $Zero\rangle$. *The model* $M_1$ *says that each* `inproceedings` *should have* 0 *occurrence of the* `number` *element. In Figure 1, the second and the third* `inproceedingss` *satisfy* $M_1$*, while the first* `inproceedings` *(by Maleki and Mohades) violates it. And such an element can be identified as a* ***structural anomaly*** *with respect to* $M_1$.

Since the number of structural anomalies in big semi-structured data sets (such as DBLP) can easily be in the thousands (if not more), enumerating all 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 to concisely summarize the structural anomalies for presentation to experts, who can then distinguish true structural errors from structurally rare, but semantically correct data.

In our demonstration, VLDB conference attendees will be presented with the robust structural models and structural anomalies learned by TREESCOPE from the widely-used DBLP and Mondial data sets. They can interactively explore the process of identifying, explaining and summarizing structural anomalies in these data sets, using the TREESCOPE tool that we developed for this purpose.

## 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 will discuss each component in more detail.

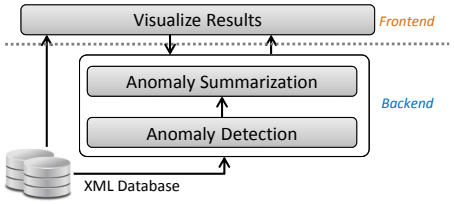


**Figure 2:** TREESCOPE System Architecture

## 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 generates a search space to explore candidate structural models and identifies structural anomalies, and then describe the **Anomaly Summarization** component.

### 2.1.1 *Structural Anomaly Detection*

**Context Path Expressions**

The first step for exploring structural models is to generate the space of context path expressions. In standard XPath, each path expression consists of a list of steps $s_1 s_2 \cdots s_m$, where $s_i =$ `axisname::nodetest[predicate]`. For simplicity we use a restricted version of XPath where `axisname` is limited to `child` and the context path expression has at most one wild card ($*$).

**Computing Frequency Distributions**

TREESCOPE learns a structural model by computing the frequency distribution $\mathcal{F}$ of the number of occurrences of the target tag $t$ in the elements in $E_c$, grouped into three buckets: $f_0(f_1, f_2, resp.)$ counts the number of elements in $E_c$ that have zero (one, two or more, resp.) occurrences of target tag $t$. This choice of bucketing is based on the cardinality quantifiers widely used in DTDs. If only one of the buckets has a non-zero count, the frequency distribution is called **consistent**. If some frequency $f_i$ in any of the buckets is smaller than an $\alpha$ fraction of the total frequency count in $\mathcal{F}$, then $\mathcal{F}$ is called $\alpha$**-skewed**. The expected frequency of a robust structural model is determined from $\alpha$-skewed frequency distributions, for a small $\alpha \in (0, 1)$. For example, if only $f_0$ is $\alpha$-skewed, the expected frequency is OneOrMore.

**Generating Lattices**

We generate a directed, acyclic search graph $G$ consisting of possible context path expressions as graph vertices.[1] For each target tag $t$, we select a subgraph $G_t$ of $G$ to explore, and compute the frequency distribution $\mathcal{F}$ for $t$ given each context path expression $c \in G_t$ – this generates the space of candidate structural models for $t$, and we repeat this process for every tag.

We seed the graph with a vertex (i.e., a context path expression $c$) that selects only the root node of the XML data tree; in DBLP this is `/dblp`. We create new graph vertices $v'$ by modifying $c = s_1 s_2[p_2] \cdots s_m[p_m]$ of an existing vertex $v$ in one of the following three ways:

**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 `child::*` in a `nodetest` of $c$ by an element name `elemName`, or (ii) adding or conjoining

[1]We use vertices to refer to the search graph, to distinguish them from nodes that are used to refer to the XML data tree.

an atomic predicate of the form `child::*` (if $c$ does not already contain a wild card), `child::elemName` or the negated predicate `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 `child::elemName` in a `nodetest` of $c$ by the wild card (*), if $c$ does not already 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 `child::*` to $c$, if $c$ does not already 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. Further, if there is a directed edge from $v$ to $v'$ in a lattice, the set of elements $E_{c'}$ 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. To speed up, TREESCOPE stops expanding the structural model when either it is **consistent** or is has **insufficient support**($|E_c| \leq \theta$), as a consistent model will never lead to any anomalies and a lowly supported model is not robust enough.

**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.[2] 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 the number of anomalies could be in 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 hence is modeled as a weighted set cover problem to cover all the structural anomalies using a minimal number of structural models. This can be solved using the standard greedy weighted set cover heuristic, which iteratively favors sets with the highest value of marginal benefit per unit cost.

## 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 `inproceedings` with child `pages`, without any `author` elements, as the expected frequency of authors is OneOrMore.

For the users who want to dig deeper, TREESCOPE will present the lattice structure, as shown in Figure 4, where 4) the left panel

[2]Note that such leaves may have outgoing vertical edges.

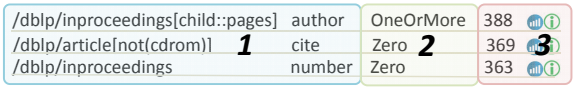


**Figure 3:** Summary of Structural Anomalies

shows the distribution under selected context path expression $c$ and 5) the right panel shows the circular layout for lattice, where each vertex is colored according to its expected frequency: blue(Zero), yellow(One), red($OneOrMore$), etc. The more colorful a lattice is, the more heterogeneous the set of expected frequencies is.

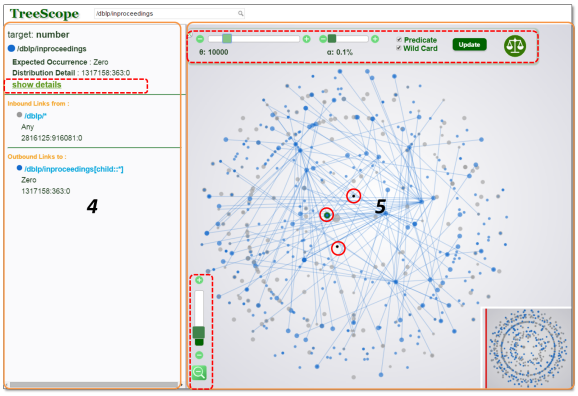


**Figure 4:** Detailed View of Lattice

Further the users can gain more insights by using any of the widgets: 1) Show Detail: retrieve sample data in different frequency buckets; 2) Parameters Slider: by adjusting the $\alpha$ and $\theta$ slider bars, the users can check the difference in anomalies detected; 3) Zoom in/out: the user can tune vertical slide bar for a more clear view, if she is only interested in a small portion of it.

# 3. DEMONSTRATION DESCRIPTION

## 3.1 Effectiveness Study

We design a baseline where the context path expression in the structural model is restricted to a single step, and neither predicate nor wild card is used. The experiment is performed against two datasets: the DBLP data published in 2013 and the Mondial data published in 2009. The former is flat in structure but large in size, while the latter is deep in structure but small in size. We evaluate for each dataset the anomalies detected by both algorithms, and take the union of the true positives as the ground truth. Then we compute the precision and recall of the two algorithms, as shown in Table 1.

**Table 1:** Comparison with Baseline

| | DBLP | | Mondial | |
|---|---|---|---|---|
| | Baseline | TreeScope | Baseline | TreeScope |
| **precision(%)** | 88 | 81 | 52 | 76 |
| **recall(%)** | 64 | 100 | 51 | 100 |
| **#anomaly** | 5,255 | 9,006 | 149 | 199 |

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 looses a little in precision on DBLP dataset, since TREESCOPE detects more anomalies on `cdrom` and `note`, which are assumed as rarity rather than true errors here.

Users can try to play with different threshold settings to get a more ideal result. When the thresholds $\theta$ and $\alpha$ vary, the list of anomalies detected changes. To achieve a relatively better result, the user can tune the frequency threshold $\theta$ and skew threshold $\alpha$ through the slider bars. A larger $\theta$ reduces the number of vertices in the lattice, while a larger $\alpha$ may result in a larger portion of elements identified as anomalies. If the user is satisfied with the default setting, but wants to see more potential anomalies, she may lower the $\theta$ value and increase the $\alpha$ value. On the other hand, if many anomalies reported are actually false positives, the user should try a smaller $\alpha$ for a more restricted skewness condition and increase $\theta$ to filter out less frequent elements.

## 3.2 Demo Walkthrough

The TREESCOPE system is deployed on our server. [3] In the demonstration, users may explore the structural anomalies from both DBLP and Mondial datasets.

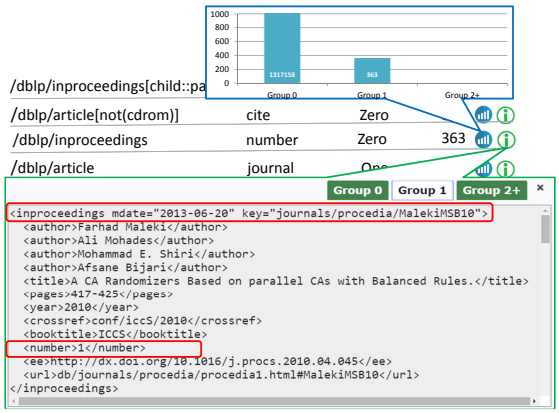


**Figure 5:** Frequency Distribution and Sample Data

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 what the anomalies exactly are, 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 conference papers as they are 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 can 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, and 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. Therefore users will see clearly the coverage of each vertex, and see if there are any interesting vertices missing from the summarization.

TREESCOPE also permits predicates and a wild card in the context path expression. The user can find the difference in generated lattices and detected anomalies by switching on/off the predicate and wild card functions from our visualization tool, and comparing them side by side. Due to limited space, we will not present details in this work.

[3] `http://42.61.39.87/TreeScope/`

# 4. RELATED WORK

Recently several approaches have been proposed for scheme inference on XML documents. XTRACT [5, 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 to be not only correct but also fairly comprehensive. As a consequence, this approach is also not practically viable for automatically identifying structural errors in semi-structured data.

# 5. CONCLUSION

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.

# 6. REFERENCES


[1] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Data-Centric Systems and Applications. Springer, 2006.

[2] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of Concise DTDs from XML Data. In *VLDB*, pages 115–126, 2006.

[3] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[4] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *SIGMOD*, pages 165–176, 2000.

[5] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. DTD Inference From XML Documents: The XTRACT Approach. *IEEE Data Eng. Bull*, pages 19–25, 2003.

[6] S. Grijzenhout and M. Marx. The Quality of the XML Web. In *CIKM*, pages 1719–1724, 2011.

[7] M. Ley. DBLP - Some Lessons Learned. *PVLDB*, 2(2):1493–1500, 2009.

[8] J. Sankey and R. K. Wong. Structural Inference for Semistructured Data. In *CIKM*, pages 159–166, 2001.

[9] B. Q. Truong, S. S. Bhowmick, C. E. Dyreson, and A. Sun. MESSIAH: Missing Element-conscious SLCA Nodes Search in XML Data. In *SIGMOD*, pages 37–48, 2013.