Inferring Structure in Semistructured Data

SVETLOZAR NESTOROV* SERGE ABITEBOUL† RAJEEV MOTWANI‡

Department of Computer Science
Stanford University
Stanford, CA 94305-9040
{evtimov,abitebou}@db.stanford.edu, rajeev@cs.stanford.edu

Abstract

When dealing with semistructured data such as that available on the Web, it becomes important to infer the inherent structure, both for the user (e.g., to facilitate querying) and for the system (e.g., to optimize access). In this paper, we consider the problem of identifying some underlying structure in large collections of semistructured data. Since we expect the data to be fairly irregular, this structure consists of an approximate classification of objects into a hierarchical collection of types. We propose a notion of a type hierarchy for such data, and outline a method for deriving the type hierarchy, and rules for assigning types to data elements.

1 Introduction

An increasing number of information sources available to the casual user export data in a variety of different formats. In most cases, the data has some structure but it is too irregular to be easily modeled using a relational [14] or an object-oriented approach [7]. We refer to this as semistructured data. Discussions of semistructured data have recently appeared in the literature [1, 4]. Because of the very nature of semistructured data, it becomes important to derive a concise representation or a summary of the inherent structure in order to give the casual user some idea of the structure and contents of the data source. Such information facilitates query formulation and can also be used for query optimization. We outline a method for inferring some underlying structure, more precisely, an approximate classification of objects into types, for large collections of semistructured data.

Several approaches have been proposed recently [5, 10, 13] to describe the "schema" of a semistructured database using graphs. In one approach [5], the schema is assumed to be given a priori. However, notably for Web data, the schema is rarely given a priori. In another approach [10, 13], it is required that the schema be a faithful representation of the data. For large and irregular data sets, such a schema may become very complex and difficult to use. Our goal is to extract a "reasonably small approximation" of the typing of a large and irregular data collection.

Following two recent independent proposals [6, 12], we assume that the data consists of a directed labeled graph. For a concrete example, consider the integration of several data sources containing information about movies found on the Web. We assume that the data is "wrapped" in a common model, specifically OEM, as done in Tsimmis [8]. In this model, the data is represented as a labeled graph of objects where the labels stand for relationships between objects. Because the data is drawn from many different sources, it is relatively irregular. To obtain a concrete sense of the kinds of problems we wish to address, suppose that the resulting database consists of thousands of labels and hundreds of thousands of objects, most of which have relatively few (dozens) of distinct labels on outgoing edges. Consider now a browser or a QBE-like interface for such a data set. The user will rapidly be overwhelmed by the sheer number of alternative labels to choose from. Thus, it is important to be able to automatically analyze the data, type the objects (to the extent possible), assign meaningful names to

---

* Supported by a grant from IBM, a gift from Hitachi, and MITRE agreement number 21263.

† Permanent address: INRIA-Rocquencourt, 78153 Le Chesnay, France

‡ Supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

these types, distinguish the "core" attributes from the more circumstantial ones, etc. We do not expect a precise and complete description of the database since the data may be too irregular. However, the techniques should be able to adapt to the needs of user, e.g., to refine the description locally if so desired, perhaps explain the precision of the typing that is obtained, or the degree of irregularity of the data.

In this paper, we propose a notion of a type hierarchy for semistructured data, an algorithm for deriving the type hierarchy, and rules for assigning types to objects of semistructured data collections. Our initial idea was to employ data mining techniques developed for mining association rules [2]. Clearly, other techniques developed in the areas of machine learning, classification and clustering, e.g., [11, 9], are relevant to a certain extent and could provide alternative approaches. After running some experiments with association-rule mining techniques, we found the results somewhat unsatisfactory. The notions of *support* and *confidence* that are central to mining association rules seem less pertinent to our problem. Instead, we propose a technique based on another criteria, called *jump*, that captures the relative importance of some attributes in a larger set. We propose an algorithm to select types and assign objects to types. As previously mentioned, we do not insist that this provide a high-precision typing of the data. In particular, some objects may remain untyped and other objects may be assigned a type that does not describe them exactly.

We briefly discuss some preliminary experimental results on Web data. While our initial results are encouraging, more experiments are needed, particularly with larger data sets. Comparison with more standard techniques such as BDDs [3] should also be performed. Finally, our initial experiments allowed us to further refine our algorithm. This paper describes on-going research. We are currently working on improving the technique and the performance of the algorithm.

2 Preliminaries

In this section we describe the data model and define some terminology that is needed for the next section. Two similar models for semistructured data have been proposed recently and independently [6, 12]. In both models, semistructured data is modeled as a rooted, labeled, directed graph with the objects as vertices and labels on edges. While we will employ the Object Exchange Model (OEM) [12], our work is equally applicable to any graph-based data model (e.g., [6]). An example of an OEM database is shown in Figure 1..

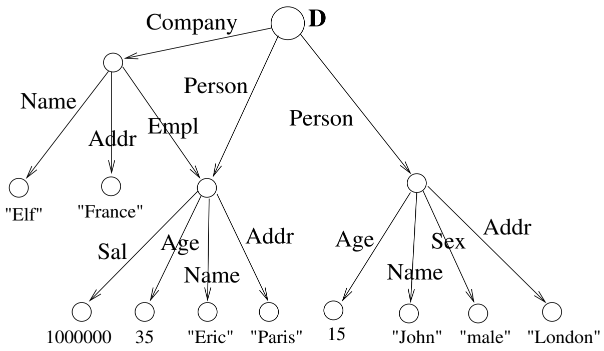

Figure 1: Part of the example OEM database D.

Let D denote the data set. For each object o in D, let $\texttt{attributes}(o)$ be the set of labels on the outgoing edges at o, and let $\texttt{roles}(o)$ be the set of labels on incoming edges at o. For a set S of labels and a data set D, we define $\texttt{at}(S)$ to be the number of objects o in D such that $\texttt{attributes}(o) = S$, and $\texttt{above}(S)$ to be the number of objects o in D such that $\texttt{attributes}(o) \supseteq S$. Note that $\texttt{above}(S) \geq \texttt{at}(S)$ because all objects counted in $\texttt{at}(S)$ are also counted in $\texttt{above}(S)$. We also define the following function: for each S,

$$\texttt{jump}(S) = \frac{\texttt{at}(S)}{\texttt{above}(S)},$$

where $\texttt{jump}(S)$ is set to 0 whenever $\texttt{at}(S) = 0$, regardless of whether $\texttt{above}(S)$ is 0 or not. Since for any S and D, $0 \leq \texttt{at}(S) \leq \texttt{above}(S)$, then we have $0 \leq \texttt{jump}(S) \leq 1$.

3 Algorithm

In this section we present an algorithm for constructing a *type hierarchy* for a semistructured data source. We also present the rules for assigning types to objects given a type hierarchy. The skeleton of our algorithm consists of four main steps, some of which may be applied iteratively.

Step 1: Identify *candidate* types.

Step 2: Select *types* from the candidates and organize them into a *type hierarchy*.

Step 3: Derive the *typing rules*.

Step 4: Validate or *type-check* the type hierarchy against the data.

For ease of exposition, we use a small and rather simplistic example to introduce the algorithm. The basic idea is to use *jumps* to discover the types, i.e., the increase in the number of "fitted" objects when an attribute is added to a set. Besides this guiding principle, the choice of types and the assignment of types to objects is based on a number of heuristic rules. The rules in a real system should be expected to be more complicated[1] than those presented in the paper. We focus here on the main idea and mention briefly possible improvements.

Our running example is a data set D that contains various information about people and companies such as their names (for both companies and people), addresses (for both companies and people), age (for people), sex (for people), salary (for people), employees (for companies), and subsidiaries (for companies). We will illustrate how our algorithm derives a *type hierarchy* for D which is intuitively correct in this simplistic example.

3.1 Identifying Candidate Types

The types we consider are characterized by sets of labels. Intuitively, an object o has type τ if the set of labels on edges with source o coincide with τ. Of course, this is too demanding so we will insist that this set be as close as possible to τ.

To identify *candidate types*, we first create a *counting lattice*, L, with an alphabet consisting of all distinct labels in D. The counted words are $\texttt{attributes}(o)$ for all objects o in D. Note that from L we can efficiently compute the functions `at` and `above` for every set of labels. The counting lattice L can be constructed in one pass over D, as can the computation of `at` values. The task of computing `above` values can be performed in time $O(n^2)$, where n is the number of non-zero `at` values, which should be significantly less than than the size of D in any reasonable application.

To continue with our example, suppose the relevant part of L (i.e., with non-zero `at` values) for the data set D is as shown in Figure 2. Each vertex contains the lattice entry (i.e., the set of different labels) associated with the vertex and the number of exact occurrences of the word, i.e., the `at` value. For example, the bottom vertex corresponds to the fact that in D there are 100 objects that have only subobjects labeled `Name` and `Addr`.

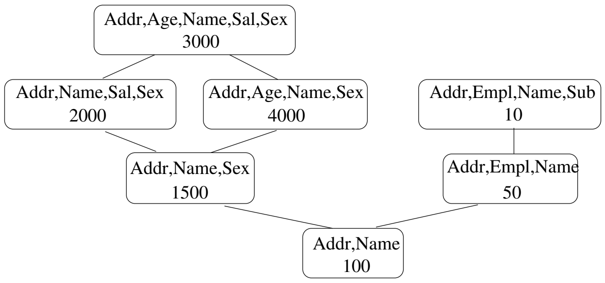

Figure 2: The counting lattice L constructed from the database D.

Once L has been created, we identify sets of labels (i.e., vertices in the lattice) that present *significant jumps* by selecting all sets of labels S such that $\texttt{jump}(S) \geq \theta$, where θ is a predetermined threshold. (The choice of θ will be discussed later.) The sets of labels with significant jumps are added to the set of candidate types. Then, we try to obtain more vertices with significant jumps by pushing counts "down" for vertices that are not "above" any candidate type. Intuitively, if an object is not in a candidate type, it is going to be assigned to less precise types, thereby increasing the population of such types and possibly turning them into candidates. Clearly, more complex rules may be used here; for instance, we could decide to attach some objects to a type with more attributes than what they actually have, thereby growing the population of more refined types.

To continue with our example, suppose that we choose a threshold $\theta = 0.7$. Then, there are three significant jumps in the lattice L shown in Figure 2:

- $\texttt{jump}(\{\texttt{Addr}, \texttt{Age}, \texttt{Name}, \texttt{Sal}, \texttt{Sex}\}) = 1$
- $\texttt{jump}(\{\texttt{Addr}, \texttt{Empl}, \texttt{Name}, \texttt{Sub}\}) = 1$
- $\texttt{jump}(\{\texttt{Addr}, \texttt{Empl}, \texttt{Name}\}) = 0.83$

After pushing counts down we get an additional significant jump as the `at` value of $\{\texttt{Addr}, \texttt{Name}, \texttt{Sex}\}$ is incremented to 7500 and we obtain that

$$\texttt{jump}(\{\texttt{Addr}, \texttt{Name}, \texttt{Sex}\}) = 0.71.$$

[1] Indeed, our prototype does use more complex rules.

Remark 3.1 *As mentioned in the introduction, we first intended to use an approach involving data mining for association rules [2]. However, looking for types with large support, i.e., large* at *values, does not work. This would lead to missing some types that occur relatively infrequently even though they are rather regular in terms of their attributes and neatly distinguished from the rest of the data.*

3.2 Building the Type Hierarchy

In the first step, we focused exclusively on the attribute labels of each object. Here we also consider role labels. Simplifying the problem for exposition purposes, for each of the candidate types, we define its *primary role* as the label occurring most frequently in $\mathtt{roles}(o)$ for all objects o of the given candidate type. We will denote the primary role of a candidate type S as $\mathtt{p}\text{-}role(S)$. Then we select candidate T as a *type* if there does not exist another candidate T' such that $T' \subset T$ and $\mathtt{p}\text{-}role(T) = \mathtt{p}\text{-}role(T')$. Intuitively, we choose the minimal set of attributes necessary to distinguish a type.

Going back to our running example, Figure 3 shows the candidate types chosen from Figure 2 and their primary roles. We find three types, namely $\{\mathtt{Addr}, \mathtt{Name}, \mathtt{Sex}\}$ with a primary role *Person*, $\{\mathtt{Name}, \mathtt{Addr}, \mathtt{Empl}\}$ with a primary role *Company* and $\{\mathtt{Name}, \mathtt{Addr}, \mathtt{Age}, \mathtt{Sex}, \mathtt{Sal}\}$ with a primary role *Empl*. Note that the candidate $\{\mathtt{Addr}, \mathtt{Empl}, \mathtt{Name}, \mathtt{Sub}\}$ does not become a type because it is a superset of $\{\mathtt{Addr}, \mathtt{Empl}, \mathtt{Name}\}$ and their primary roles are the same.

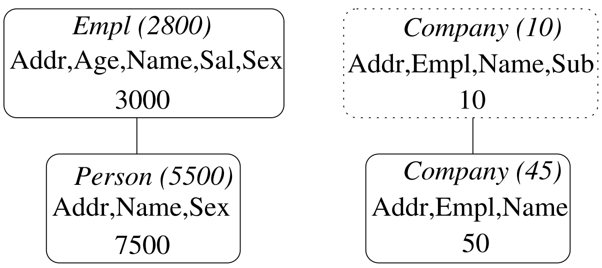

Figure 3: The candidate types with their primary roles.

In general, we can use more than one label as the primary role of a given candidate S. Indeed, it might be the case that the two most frequent labels in $\mathtt{p}\text{-}role(S)$ occur an (almost) equal number of times. In our simple example we do not address this problem but our algorithm can handle more complex structures in $\mathtt{p}\text{-}role(S)$, e.g., a set of labels with weights. In this case, the rules for choosing the types from the candidates become more complex.

3.3 Typing Rules

Let the types we found in the previous step be $S_1, \ldots, S_N$. Consider an object o. Then we assign to o the type S_k that has the shortest "distance" to o. By distance[2] we mean the number of labels in the set differences $\mathtt{attributes}(o) - S_k$ and $S_k - \mathtt{attributes}(o)$.

Note that a given object may be assigned to more than one type. We consider this to be a feature of the algorithm rather than a shortcoming. In many real life situations, objects do belong to more than one type.

3.4 Validation and Evaluation

Once we have build the type hierarchy and assigned types to the objects, we need to evaluate the result and validate the classification we obtained. One important measure is the *type size* (e.g., the number of classes) of the typing. Another category of measures involves *correctness* or *accuracy* of the typing. Consider, for instance, the number of objects that have been assigned a certain type even though they are missing some of the attributes characterizing the type or they have more than what is required. Also, consider the number of objects that we failed to classify.

As mentioned earlier, the result of the algorithm depends crucially on the choice of threshold θ that we considered so far somewhat arbitrary. Clearly, there is a trade-off between type size and accuracy. For example, with $\theta = 0$, we obtain a perfect typing by creating a separate type for each slight variation of object structure. On the other hand, a too high θ would yield very few types and thus may result in very low accuracy. If the number of classes does not fit our expectations (e.g., is too large to be tractable) or if the accuracy is not sufficient, we have to try new values for θ.

It would be useful to relate directly θ to the database size, number of labels, type size, accuracy and other fixed parameters of the problem. However,

[2] More complex distance measures could clearly be used, e.g., a distance that would give less weight to the presence of an extra attribute than to the absence of a required one.

this is ignoring another important measure - the *degree of regularity* of the data set. This degree of regularity may be a useful information for the system (e.g., for physically organizing the data) as well as for the user who is told what kind of data to expect. It can also be useful in guiding the choice of a value for θ.

4 Conclusions

We outlined an algorithm for deriving a type hierarchy for a semistructured data source and rules for assigning types to objects. The algorithm evolved from experiments on Web data. In our experiments, we used two different data sources: a subset of the ESPN SportsZone[3] that provides various sports information, and an on-line database containing information about the Stanford Database Group (DBG)[4]. Both data sets are of relatively modest size (hundreds of objects and dozens of labels) but DBG is highly cyclic whereas ESPN is close to a tree. The typing is simple enough so we could interpret the results of the algorithm; but the data is irregular enough so that finding the type hierarchy is nontrivial. Our initial results are encouraging although clearly more experiments and work are needed. In particular, as expected, our algorithm is sensitive to the *jump threshold* in the sense that lower threshold values result in a greater number of types. We plan to investigate techniques to provide "good" estimates for this threshold. Also, the rules that we present in this paper are simplified for presentation purposes. The choice of such rules has a strong impact on the quality of the results, and we are currently experimenting with more complex rules and working on the fine tuning of our algorithm with respect to such rules. Finally, we designed the algorithm with performance in mind. We are now working on designing appropriate access structures to improve the performance of our prototype.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proceedings of ICDT*, pages 1–18, Delphi, Greece, January 1997.

[2] R. Agrawal, T. Imilienski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.

[3] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[4] P. Buneman. Semistructured data: a tutorial. In *Proceedings of PODS*, pages 117–121, Tuscon, Arizona, May 1997.

[5] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Addind structure to unstructured data. In *Proceedings of ICDT*, pages 336–350, Delphi, Greece, January 1997.

[6] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference*, pages 505–516, Montreal, Canada, June 1996.

[7] R.G.G. Cattell. *Object data management*. Addison-Wesley, Reading, Mass., 1994.

[8] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The Tsimmis project: Integration of heterogeneous information sources. In *Proceedings of 100th Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, October 1994.

[9] D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors. *Machine learning, neural and statistical classification*. Prentice Hall, Englewood Cliffs, N.J., 1994.

[10] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In *Proceedings of ICDE*, pages 79–90, Birmingham, U.K., April 1997.

[11] N.J. Nilsson. *The mathematical foundations of learning machines*. Morgan Kaufmann, San Mateo, Calif., 1990.

[12] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of ICDE*, pages 251–260, Taipei, Taiwan, March 1995.

[13] D. Quass *et. al.* Lore: A lightweight object repository for semistructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 549, Montreal, Canada, June 1996.

[14] J.D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, Rockville, Maryland, 1989.

---

[3] http://espnet.sportszone.com/

[4] http://www-lore.stanford.edu:8765/ui2/