Schema Discovery for Semistructured Data

Ke Wang **Huiqing Liu**
Department of Information Systems and Computer Science
National University of Singapore
wangk@iscs.nus.sg, liuhuiqi@iscs.nus.sg

Abstract

To formulate a meaningful query on semistructured data, such on the Web, that matches some of the source's structure, we need first to discover something about how the information is represented in the source. This is referred to as *schema discovery* and was considered for a single object recently. In the case of multiple objects, the task of schema discovery is to identify typical structuring information of those objects as a whole. We motivate the schema discovery in this general setting and propose a framework and algorithm for it. We apply the framework to a real Web database, the Internet Movies Database, to discover typical schema of most voted movies.

Introduction

As the amount of data available on-line grows rapidly, we find that more and more of the data is *semistructured*. In the semistructured world, data has no absolute schema fixed in advance, and the structure of data may be irregular or incomplete. Semistructured data arise when the source does not impose a rigid structure (such as the Web) and when data is combined from several heterogeneous data sources. See (?) for an excellent survey on semistructured data. Unlike traditional relational or object-oriented databases where an external schema is known in advance, semistructured data is self-describing in that each object contains its own schema, and the distinction of schema and data is blurred.

[1]

Figure 1 shows a segment of the information about the top soccer league (The Premiership) in England. Each circle plus the text inside it represents an object and its identifier. The arrows and their labels represent object references and semantics, which are the main structuring information of objects. There is a large degree of irregularity: different clubs may have a different number of players; some fields may be missing for some clubs; some players have first and last names separately recorded; some just have a single full name recorded; some have nicknames; etc. Other examples of semistructured data are LaTex and BibTex files, genome databases, drug and chemical structures, scientific databases, libraries of programs, production schedules, task definitions and more generally, digital libraries, on-line documentations, electronic commerce.

To formulate any meaningful query on semistructured data, such as on the Web, that matches some of the source's structure, we need first to discover something about how the information is represented in the source, called the *data guide* in (?). This is referred to as *schema discovery* in (?). Schema discovery explores an object by moving (navigating) from an object to its subobjects and keeping track of the labels of the object references traversed. For example, schema discovery may find that *persons* possibly have outgoing edges labelled *name*, *address*, *hobby* and *friend*, that an *address* is either a string or an object having outgoing edges labelled *street* and *zipcode*. Such information is very helpful for query specification, query processing and optimization, data organization, and search strategies. We quote (?) here for the gist of the motivation:

> ⋯ More generally, one could envision the use of general purpose data mining tools to extract structuring information. One can then use the information extracted from the files to build a structured layer above the layer of more unformed data. This structured layer references the lower data layer and yields a flexible and efficient access to the information in the lower layer to provide the benefits of standard database access methods. ⋯

In this paper, we consider a modified problem with the same motivation. We are interested in structuring

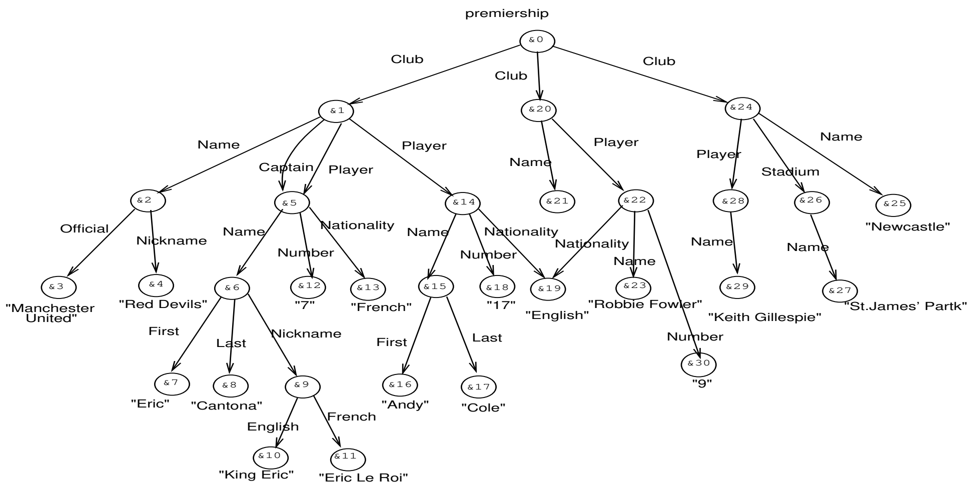

Figure 1: The premiership object

information that is typical of a majority of a given collection of objects. Such information are called *schema patterns*. The main difference from (?) is that we consider multiple objects, rather than a single object, for the discovery task. The motivation is simple: it is very common for an application to deal with a collection of objects, such as finding the typical structure of a collection of movie documents. As suggested above, the discovered schema patterns can be used to build a structured layer, e.g., relational or object-oriented databases, over semistructured objects.

Schema Discovery

We adopt the Object Exchange Model (OEM) (?) that represents semistructured data by a labelled graph.

The object-exchange model (OEM). OEM is a simple, self-describing object model with nesting and identity. Every object in OEM consists of an *identifier* and a *value*. The *identifier* uniquely identifies the object. The *value* is either an atomic quantity, such as an integer or a string, or a bag of object references, denoted as $\{l_1 : id_1, \ldots, l_p : id_p\}$, where l_i are labels and id_i are identifiers of subobjects. The label describes the meaning of the relationship between the object and its subobject. We say that id_i are *labelled* by l_i. We assume that $f(id)$ returns the bag of object references for bag object id. (Often, we don't distinguish between an object and the identifier of the object.)

We can view OEM as a graph where the nodes are the objects and the labels are on the edges (object references). An OEM is *acyclic* if the graph is acyclic. We consider only acyclic OEM. The bag semantics allows duplicates, thus, generalizes the set semantics. For example, using the bag semantics we can model the structuring information which tells that there are at least 10 players in 90% of clubs.

A transaction database is a collection of complex objects on which schema discovery is performed. These objects are called *transactions*. In the early example, a transaction database can be a collection of club objects if we wish to perform schema discovery on them. We make use of two special symbols. ? denotes the wildcard label that stands for any label. $\perp$ denotes the nil schema that contains no schema information. The following notion generalizes simple path expressions in (?).

$\perp$ is a *tree expression* of every object. Assume that te_i are tree expressions of objects id_i, $1 \leq i \leq p$, and that $f(id) = \{l_1 : id_1, \ldots, l_p : id_p\}$. Then $\{l_{i'_1} : te_{i_1}, \ldots, l'_{i_k} : te_{i_k}\}$ is a *tree expression* of object id, $k > 0$, where either $l_{i'_j} = l_{i_j}$ or $l_{i'_j} =?$. Intuitively, a tree expression of an object is a partial (labelled) tree representation of the object, in which identifiers at nodes are ignored and labels on edges may be replaced with wildcard ?. A tree expression of an object "generalizes" the object by containing less information. If a tree expression is shared by a significant number of transactions, the tree expression represents a structuring pattern of these transactions.

Example 1 *Consider the transaction database* $\{\&1, \&20, \&24\}$ *for three clubs in Figure 1.* $te_1 = \{Player : \{Name : \perp\}, Name : \perp\}$ *is a tree expression of* &1, &20, &24*, so is the result of replacing any label in* te_1 *with ?.* $te_2 = \{Player : \{Name : \perp, Nationaltiy : \perp\}, Name : \perp\}$ *is a tree expression of* &1 *and* &20*, but not of* &24*.*

We can compare the generalization power of tree expressions by the "weaker than" relation defined below. $\perp$ is *weaker than* every tree expression. A tree expression $\{l_1 : te_1, \ldots, l_p : te_p\}$ is *weaker than* tree expression $\{l'_1 : te'_1, \ldots, l'_q : te'_q\}$ if for $1 \leq i \leq p$, te_i is weaker than some te'_{j_i} such that either $l_i = l_{j_i}$ or $l_i = ?$, where j_i are different for different i.

Definition 1 (Schema discovery) *Consider a transaction database. Let te be a tree expression of some transaction. The* support *of te is the number of transactions t such that te is a tree expression of t. For a user-specified minimum support $MINISUP$, te is* frequent *if the support of te is not less than $MINISUP$. te is* maximally frequent *if te is frequent and is not weaker than other frequent tree expressions. te is a* schema pattern *or simply* pattern *if te is maximally frequent. The schema discovery problem is to find all patterns.*

Example 2 *Continue with Example 1. The support of te_1 is 3 and the support of te_2 is 2. Therefore, if $MINISUP = 3$, te_1 is frequent, but te_2 is not. Suppose $MINISUP = 2$. Both te_1 and te_2 are frequent. Let $te_3 = \{Player : \{Name : \perp, Nationality : \perp, Number : \perp\}, Name : \perp\}$. te_3 is supported by* &1 *and* &20*. Since te_1 and te_2 are weaker than te_3, they are not patterns. For the same reason, nor is the result of replacing any label in te_3 with wildcard ?. In addition, te_3 is not weaker than any frequent tree expression except itself. Therefore, te_3 is a pattern, which captures the maximal common structure of supporting transactions* &1 *and* &20*.*

The algorithm

To compute patterns in the increasing size of tree expressions, we define a *k-tree expression* to be a tree expression containing k occurrences of $\perp$. Each $\perp$ occurrence in a k-tree expression corresponds to a special path in G, called a G-path. A G-path is a sequence of alternating objects and labels that starts with the dummy transaction $\top$ and ends with a node in G. Repeating occurrences of the same $label : id$ pair are represented by superscripted $label : id^1, \ldots, label : id^k$. A k-tree expression is represented by a sequence $p_1 \ldots p_k$, where p_i is the G-path for the ith $\perp$ occurrence.

Alternatively, a sequence of G-paths $p_1, \ldots, p_k$ can be represented by a *label tree* obtained by "assembling" the G-paths in a natural way. Figure 2 shows that three label trees for three sequences $p_1p_2, p_1p_3, p_1p_2p_3$ of G-paths, where $p_1 = \{\top, l_1 : a^1, l_2 : 1^1\}$, $p_2 = \{\top, l_1 : a^1, l_2 : 1^2\}$, and $p_3 = \{\top, l_1 : a^1, l_2 : 1^3\}$. Identifiers and their superscripts are significant in constructing a label tree, but are ignored after the tree is constructed. We will represent a k-tree expression interchangeably by a sequence of k G-paths or the corresponding label tree.

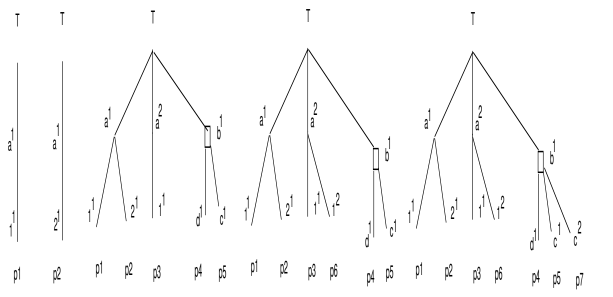

Figure 2: Constructing $p_1p_2p_3$ by extending p_1p_2 by p_1p_3

The algorithm makes multiple passes over the transaction file D to compute the set F_k of frequent k-tree expressions, $k = 1, 2, \ldots$ In the first pass, we find all frequent 1-tree expressions, F_1, in the form of G-paths. Each subsequent pass over D computes F_k from F_{k-1}. The following theorem forms the basis of computing F_1.

Theorem 1 *Let p_i be G-paths. Every frequent k-tree expression $p_1 \ldots p_{k-1}p_k$ is constructed by two frequent $(k-1)$-tree expressions $p_1 \ldots p_{k-2}p_{k-1}$ and $p_1 \ldots p_{k-2}p_k$ such that p_{k-1} is not a prefix of p_k and vice versa.*

Theorem 1 gives a superset of F_k, called *k-candidates*. The actual frequent k-tree expressions in F_k are found by counting the support of k-candidates during a pass over the transaction file D. It does not work to simply treat G-paths as items and find schema patterns as large itemsets in (?) because the connectivity among nodes on G-paths is important.

To reduce the cost of search, the storage structure of F_{k-1} must facilitate efficient retrieval of the matching pairs in Theorem 1 and dynamically extend from F_{k-1} to F_k without reorganization. In addition, heuristics of pruning useless search space are crucial to the performance. Due to space limitation, we omit the detail of the algorithm.

The maximal phase

After all F_k's are computed, we prune all non-maximally frequent tree expressions. An important observation is that, for $i > j$, no i-tree expression can be weaker than a j-tree expression. Therefore, the pruning is done in the order $F_k, F_{k-1}, \ldots, F_1$.

Experiments

We tested the proposed discovery framework on the Internet Movies Database (IMDb) on the Web at http://us.imdb.com. IMDb currently covers more

```
Pattern 1 (support = 22%):
{Title, Released_Year, Country, Award, Key, Distributor,
 Director:{Name, Place, Award, Spouse:{Name}, Category},
 Cast:{{Name, Place, Award, Spouse:{Name, Occupation}, Category},
       {Name, Place, Spouse:{Name, Occupation}, Category},
       {Name, Place, Spouse:{Name}, Category},
       {Name, Place, Spouse:{Name}},
       {Name, Place}},
 Writer:{Name, Place, Category},
 Composer:{Name, Category},
 Cinematographer:{Name, Category},
 Editor:{Name},
 Producer:{Name, Place, Category}}.
```

Figure 3: A sample pattern

than 95,000 movies and over 1,300,000 filmography entries. All movies are organized into HTML document trees. To make the manual verification easier, we chosen only the top 100 movies from the entry Top 250 movies as voted by YOU (http:www.us.imdb.com/top_250_films) for the transaction database, and picked only the top five actors in each movie. The OEM is constructed by extracting from the HTML documents data items of the form $l : v$, where l is a label in bold face and v is either a hyperlink (i.e., a complex object) or an atomic object. The extraction ignores raw data such images and free English text. We set $MINISUP$ to 15%.

The first pattern found is given in Figure 3. ($\perp$ is omitted in all patterns, so is the label *actor*.) This pattern tells, among others, that 22% of movies share the structure in the pattern. For example, it tells that one actor got award, four actors had spouses, two of which were doing movie related jobs, etc.

The second pattern with support = 17% is the same as the first pattern except that the last four lines are replaced by

```
?:{Name, Place, Spouse:{Name}, Category}}.
```

The wildcard label ? is for anyone who is a producer, a writer, a editor, a cinematographer, etc. This substructure was not discovered in the first pattern because it is not frequent enough if the wildcard ? is not used. Since Director is in the second pattern, ? does not include Director unless a movie has more than one director.

Third pattern with support = 16% is the same as the first pattern except that the line for Director is deleted and the last four lines in the first pattern are replaced with

```
?:{Name, Place, Award, Spouse:{Name,
    Occupation}, Category}}.
```

The wildcard ? here covers all those covered by ? in the second pattern plus Director. This substructure was not discovered previously in the second pattern where Director was separated out.

Conclusion

We have motivated and introduced schema discovery for a collection of semistructured objects. We have addressed these issues: (a) considering multiple objects for schema discovery, (b) allowing bag construction, (c) allowing wildcard labels in patterns, (d) defining the generalization hierarchy and maximality to focus on interesting patterns, (e) defining the discovery problem and proposing an algorithm, (f) studying the effectiveness on a real Web database.

Despite the growing popularity of semistructured data, such as Web documents, KDD research has largely focused on well structured data, mainly relational tables. This paper presents one step to address this unbalance. We believe that in the semistructured world, KDD will play even a bigger role than in structured data — simply because there are more to be discovered.

References

[Abi97] S. Abiteboul, "Querying semi-structured data", ICDT 1997 (http://www-db.stanford.edu/pub/papers/icdt97.semistructured.ps)

[A+96] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J.L. Wiener, "The lorel query language for semistructured data", to appear in Journal of Digital Libraries, 1997 (http://www-db.stanford.edu/pub/papers/lorel96.ps)

[AIS93] R. Agrawal, T. Imielinski, A. Swami, "Mining association rules between sets of items in large databases", SIGMOD 1993, 207-216

[NUWC97] S. Nestorov, J. Ullman, J. Wiener, S. Chawathe, "Representative objects: concise representations of semistructured, hierarchical data", ICDE 1997 (http://www-db.stanford.edu/pub/papers/representative-object.ps)