Bulletin of the Technical Committee on

Data Engineering

September 2003 Vol. 26 No. 3

IEEE Computer Society

Letters

Special Issue on Structure Discovery

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

The Data Engineering Bulletin web page is http://www.research.microsoft.com/research/db/debull.

Nominations for Chair of TCDE

The Chair of the IEEE Computer Society Technical Committee on Data Engineering (TCDE) is elected for a two-year period. The mandate of the current Chair, Erich Neuhold, expires at the end of this year and the process of electing a Chair for the period 2004-2005 has begun. A Nominating Committee consisting of Paul Larson, Masaru Kitsuregawa and Betty Salzberg has been struck. The Nominating Committee invites nominations for the position of Chair from all members of the TCDE. To submit a nomination, please contact any member of the Nominating Committee before October 25, 2003.

More information about TCDE can be found at http://ipsi.fhg.de/tcde. Information about TC elections can be found at http://www.computer.org/tab/hnbk/electionprocedures.htm.

Paul Larson, Masaru Kitsuregawa, and Betty Salzberg
TCDE Nominating Committee

Letter from the Editor-in-Chief

The Current Issue

We database folk love our structured data. The field started as an effort to solve the business data processing problem. It is hard to imagine something more (rigidly) structured than an 80 column card having fixed length fields. Precursor database systems, called formatted file systems, were little more than cards on tape. Early database systems were liberated from the 80 column constraint by the increased flexibility of tape and disk, where records could actually be arbitrary sizes. Further liberation occurred when the database systems started to deal with variable length fields.

Despite the preceding advances, and the subsequent introduction of data independence, where one did not have to know the physical representation of data in order to query and update it, we continued to rely on complete knowledge of how data was laid out, what the attribute names and types were, indeed complete syntactic knowledge of the data. But times change. Our prior world of structured data depended upon our controlling the definition of the data and hence its structure. Today's web-ish, XML-ish world is clearly not under our control. So if the database field is to contribute to this new world, we need to meet it at least half way. One way to do this, so as to leverage our strengths in structured data, is to discover or impose structure on this new world's data.

The current issue presents a number of ways of attacking the problem of structure discovery. Given the great diversity of data, there is probably no one golden road to success in this endeavor. The current issue explores a number of interesting and widely varying approaches. Renee Miller, who herself structures her share of data, has brought together an interesting cross-section of methods for finding the structure that is in data. I want to thank Renee for the fine job she has done in producing an issue on structure discovery that should stimulate lively discussions in many parts of our field.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

In preparing this special issue on structure discovery, I was motivated by the observation that *unstructured data* is often not unstructured at all. It may possess a rich (though perhaps irregular) structure. Often, it is the absence of **known** structure rather than the absence of structure that leads to a data-set being characterized as “unstructured” or “semi-structured”. Web data and data represented in XML (a semi-structured data model) is often (though certainly not always) highly structured. Similarly, I have found in my own work on integrating and transforming structured data sources, that effective integration exploits not only the given (“imposed”) structure of a schema, but also implicit structure within the data. These observations were reinforced by an entertaining keynote by Joe Hellerstein at the recent 2003 SIGMOD WebDB Workshop that elaborated on techniques that exploit “engineered” structure in data (as done in traditional database management where a data design or schema is imposed and enforced) and those that exploit found structure (as done in information retrieval where unstructured documents are managed without a constricting schema). However, data rarely resides neatly in one of these two camps. Unstructured data may possess structure that could be exploited in its management. Similarly, the imposed schema of a structured data-set may not always do it justice as it may be incomplete or even inaccurate.

The articles in this special issue explore the question of how to find new and unknown structure in data. Notice that I have purposely not defined precisely what I mean by structure. Each article takes a different view of this, focusing on structure that can be exploited in different classes of data management applications. The articles also present techniques drawn from a variety of disciplines. It is my hope that collecting these short position statements together in one volume will illustrate the problems, permit comparisons, highlight synergies, and inspire more work in what I believe to be a very promising research direction.

Renée J. Miller
Department of Computer Science
University of Toronto

Learning and Discovering Structure in Web Pages

William W. Cohen
Center for Automated Learning & Discovery
Carnegie Mellon University
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

Abstract

Because much of the information on the web is presented in some sort of regular, repeated format, "understanding" web pages often requires recognizing and using structure, where structure is typically defined by hyperlinks between pages and HTML formatting commands within a page. We survey some of the ways in which structure within a web page can be used to help machines understand pages. Specifically, we review past research on techniques that automatically learn and discover web-page structure. These techniques are important for wrapper-learning, an important and active research area, and can be beneficial for tasks as diverse as classification of entities mentioned on the web, collaborative filtering for music, web page classification, and entity extraction from web pages.

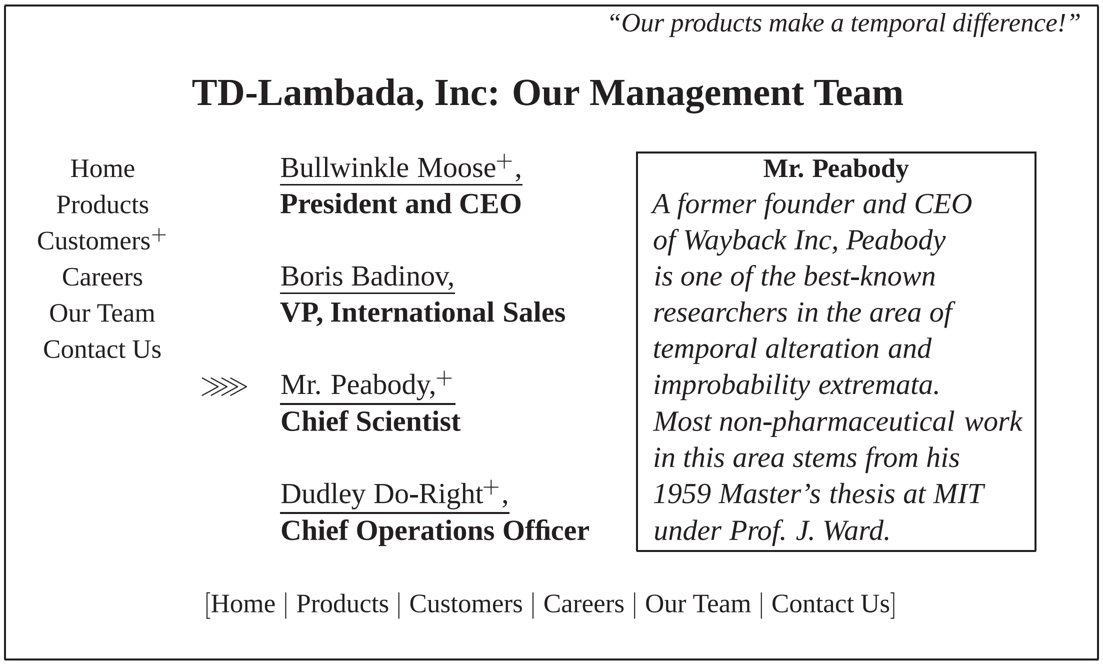
1 Introduction

In spite of recent progress on the semantic web and interchange formats like XML, most of the information available today on the world wide web is targeted at people, not machines. Because of this, the problem of automatically aggregating information from the web [3, 4, 11, 18, 21, 23, 24] is technically difficult: simply put, in order to make web information machine-readable, it is usually necessary to design a program that "understands" web pages the same way a human would—at least within some narrow domain.

The task of "understanding" web pages is broadly similar to classical problems in natural language understanding. One important difference is that most of the information on the web is not presented in smoothly flowing grammatical prose; instead, web information is typically presented in some sort of tabular, structured format. Hence understanding web pages often requires recognizing and using "structure"—a structure typically being defined by hyperlinks between pages, and HTML formatting commands (e.g., tables and lists) within a page. In this paper, we will survey some of the ways in which structure can be used to help machines understand web pages. We will focus on techniques that exploit HTML formatting structure within a page, rather than link structure between pages.

Broadly speaking, there are two ways in which such structure can be used. It may be used *directly* to accomplish some task; typically, to determine how to extract data from a page. As an example, consider the imaginary web page shown in Figure 1. (The markers near the words "Customers", "Mr. Peabody", etc will be explained below.) To populate the relation *titleOf(personName, jobTitle)* from this page, one might extract

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering


"Our products make a temporal difference!"

TD-Lambada, Inc: Our Management Team

Home
Products
Customers$^+$
Careers
Our Team
Contact Us

Bullwinkle Moose$^+$,
President and CEO

Boris Badinov,
VP, International Sales

⋙ Mr. Peabody,$^+$
Chief Scientist

Dudley Do-Right$^+$,
Chief Operations Officer

Mr. Peabody
A former founder and CEO of Wayback Inc, Peabody is one of the best-known researchers in the area of temporal alteration and improbability extremata. Most non-pharmaceutical work in this area stems from his 1959 Master's thesis at MIT under Prof. J. Ward.

[Home | Products | Customers | Careers | Our Team | Contact Us]

Figure 1: An imaginary web page containing non-trivial regular structure

names and titles from alternate non-blank positions in the second column; thus extraction would be based on the formatting used on the page.

Structure might also be used as one input among many that collectively guide a more complex process. For instance, consider using a statistical machine learning algorithm to identify job-title phrases on web pages. Such an algorithm might exploit content features (like "x contains the word 'CFO' "), formatting features (like "x is rendered in boldface") or structural features (like "x appears in the same table column as a known job-title word"). We will call this sort of use of structure *indirect*, since structure is only one of several factors that affects the learning algorithm.

In this paper, we will focus on techniques that involve learning structure from examples, or automatically discovering possible structure using heuristics. We will also discuss how learned or discovered structure can be used—both directly and indirectly.

2 Learning structure for direct use

A program that derives a database relation from a specific website is called a *wrapper*, and *wrapper learning* is the problem of learning website wrappers from examples. For instance, a wrapper for the website of Figure 1 might produce the relation *titleOf(personName, jobTitle)* described above.

As an alternative to explicit programming, such a wrapper might be learned from a handful of user-provided examples. As a very simple example, the four job titles on this page might be learned from the two examples "President and CEO" and "VP, International Sales". (Notice that these strings are *positive* examples, *i.e.*, instances of strings that *should* be extracted; most learning systems also require *negative* examples, *i.e.*, strings that *should not* be extracted. A common convention in wrapper-learning is for a user to provide *all* positive examples in some prefix of the document being labeled. A set of negative examples then can be derived by using a sort of closed world assumption.)

Wrapper learning has been an active area from the mid-nineties [20] through the present (*e.g.*, [2, 15, 25]).

Typically a wrapper is defined in terms of the format used on a particular web site: in Figure 1, for instance, a wrapper might extract as job titles “all boldfaced strings in the second column”. Hence wrapper-learning is a prototypical example of learning a structure for direct use. By way of an introduction to this active area of work, we will discuss below a few successful wrapper-learning systems.

Kushmeric’s seminal WIEN system [19, 20] was based on a handful of carefully crafted wrapper languages with very restricted expressive power. For example, one such language was HLRT. An HLRT wrapper for a one-field relation is defined by four strings: a string h that ends the “header” section of a web page, two strings ℓ and r that precede and follow each data field, and a string t that begins a “tail” section of a web page. (The “head” and “tail” sections of a page contain no data fields.) For instance, a plausible HLRT wrapper for job-title strings in an HTML version of the page of Figure 1 might be $h =$“Contact Us$\langle$/A$\rangle$”, $\ell =$“$\langle$B$\rangle$”, r=“$\langle$/B$\rangle$”, and t=“$\langle$/B$\rangle\langle$/LI$\rangle\langle$/UL$\rangle$”. By limiting a wrapper-learner to explore this restricted set of possible structures, Kushmeric was able to use a learning algorithm that was elegant and well-grounded formally (for instance, it was guaranteed to converge after a polynomial number of examples). The disadvantage of this approach was that some wrappers could not be learned at all by WIEN.

Later wrapper-learning systems such as STALKER [26] and BWI [14] used more expressive languages for wrappers: for instance, the start of a field might be identified by the disjunction of a set of relatively complex patterns, rather than a single fixed string ℓ. Surprisingly, these broader-coverage systems did *not* require more examples to learn simple structures, such as those considered by WIEN; instead the learning algorithms used were designed to propose simpler structures first, and fall back on more complex structures only if necessary. The disadvantage of these systems (relative to WIEN) is greater complexity, and weaker performance guarantees.

In our own previous work, we developed a wrapper-learning system called WL^2 [16, 9]. Like WIEN, WL^2 exploits the fact that many wrappers are based on a few simple structures, and that it is often possible to design restricted languages that capture these common cases. Like STALKER and other systems, WL^2 can find complex wrappers when simple ones are inadequate. Unlike previous wrapper-learners, WL^2 is modular, and designed to be easily extended to accomodate new classes of structures (perhaps structures common in a particular domain). The learning system in WL^2 is based on an ordered set of *builders*. Each builder B is associated with a certain restricted language $\mathcal{L}_B$. However, the builder for $\mathcal{L}_B$ is not a learning algorithm for $\mathcal{L}_B$. Instead, to facilitate implementation of new “builders”, a separate master learning algorithm handles most of the real work of learning, and the builder B need support only a small number of operations on $\mathcal{L}_B$—the principle one being to propose a single “least general” structure given a set of positive examples. Builders can also be constructed by composing other builders in certain ways. For instance, two builders for languages L_{B_1} and L_{B_2} can be combined to obtain builders for the language $(L_{B_1} \circ L_{B_2})$, or the language $(L_{B_1} \wedge L_{B_2})$.

WL^2 included builders that detected structures of various kinds. Some structures were defined in terms of string-matches, like HLRT; some were defined in terms of the HTML parse tree for a page; and some were defined in terms of the geometrical properties of the rendered page. The master learning algorithm can find extraction rules based on any of these sorts of structures; or, it can combine structures produced by different builders to form new, more complex extraction rules (e.g., “extract all table cells vertically below a cell containing the words “Job Title” that will be rendered in bold with a font size of 2”). Experiments conducted with a large number of practical extraction problems showed that WL^2 could learn to extract many data fields with three positive examples or less.

3 Discovering structure for direct use

In wrapper learning, a user gives explicit information about each structure. Even if wrapper-learning is very efficient, a system that requires information from many web sites will still be expensive to train. A natural question to ask is: can structure be recognized (and used) without training data?

In some situations, the answer is “yes”. For instance, Kushmeric *et al* [20] described ways in which automatic but imperfect entity recognizers could be used to drive wrapper learning; Embley *et al* [12] described accurate and page-independent heuristics for recognizing strings that separate one data record from another in web pages; and other researchers [13, 22] have described techniques for finding logically meaningful facts from HTML lists and tables without training data. Below we will summarize some of our own work on automatically discovering useful structure.

One difficulty with evaluating such a discovery system is that, in general, it is unclear what structures are "useful". One definition of "useful" is "structure that could be used to guide web-site wrapping". In previous work [5], we evaluated the performance of a structure-discovery algorithm by using it to propose structures for 82 previously-wrapped web pages, and measuring the fraction of the time that the discovered structure coincided with the wrapper for that page.

The algorithm proposes two types of structures, called *simple lists* and *simple hotlists*, which have the property that only a polynomial number of possible structures can appear on any given web page; this means that finding the best structure can be reduced to generating all simple lists or hotlists, and then ranking them. One simple ranking scheme is to score a structure by the number of elements extracted by the the structure; on this dataset, this heuristic ranked the correct structure highest about 20% of the time. If more information is available, then more powerful heuristics can be used. For instance, if a large list of "seed" items of the correct type are available, then ranking lists according to the an aggregate measure of the distance from extracted items to "seeds" (using an appropriate similarity metric) ranks the correct structure highest about 80% of the time.

This algorithm could thus be used as the basis for a wrapper "learning" algorithm that uses no explicit user examples. Instead wrappers would be constructed as follows:

- The system generates, ranks, and presents to the user an ordered list of possible structures. For instance, given a set of person-name seeds and the web page of Figure 1 as input, the system might produce these candidate lists:

  1. "Bullwinkle Moose", "Boris Badinov", "Mr. Peabody", ...
  2. "Bullwinkle Moose, President and CEO", "Boris Badinov VP, International Sales", ...
  3. "Home", "Products", "Customers", "Careers", ...
  4. "President and CEO", "VP, International Sales", "Chief Scientist", ...

  The ranking shown is reasonable, as structures containing words like "Mr." and "Boris" (which are likely to appear in some seed) should be ranked higher than structures containing only phrases like "Home" and "Chief Scientist".

- The user picks a candidate structure from the ranked list (say, the first one) that contains the items to be extracted.

- The user specifies semantically how these items will be stored in the database being populated; e.g., by specifying that they are the set of "people employed by TD-Lambada, Inc".

4 Discovering simple structure for indirect use

We emphasize that the 80% figure reported in the section above is relative to the set of 82 wrappers used in these experiments, all of which could actually be represented as simple lists or hotlists. Of course, many wrappers are more complex. For appropriately simple wrappers, less user intervention is required using the method described above than with than learning techniques like WL^2; however, the user is still needed to (a) verify the correctness of a proposed structure and (b) ascribe the appropriate semantics to the structure.

One way to exploit structure without involving the user at all is to use discovered structure indirectly. As an example, consider the task of classifying musical artists by genre given only their names: thus "John Williams" would be classified as "classical" and "Lucinda Williams" as "country". This is difficult to do accurately without some background knowledge about the artists involved. Such background knowledge might be obtained by manually training (and then executing) wrappers for some appropriate set of web sites—can one obtain similarly useful background knowledge using *automatically* discovered structure?

In another set of experiments [6], we used structure discovery algorithms to find features useful for learning. One task we explored was learning to classify musical artists by genre. We took a large list of musical artists and labeled a subset to use as training data. We then used the complete list of artists as seeds for (a variant of) the structure-discovery algorithm described above. Applied to a large set of web pages, this produces many possible structures. Without user filtering, many of these structures are meaningless, and other structures correspond to sets that are completely uncorrelated with genre (like "artists with names starting with 'A' that have CDs on sale at Bar.com"). However, may of the discovered structures *are* correlated with genre, and hence can be exploited by a statistical learner.

Specifically, we associated each discovered structure D with a matching subset S_D of the complete artist list. We then introduced, for every structure D, a new binary feature f_D which was true for every example $x \in S_D$ and false for every other example. Finally we applied traditional feature-based statistical learning algorithms to the augmented examples. In our experiments, the new structure-based features improved performance on several benchmark problems in a wide variety of situations, and performance improvements were sometimes dramatic: on one problem, the error rate was decreased by a factor of ten. Similar features can be used for other tasks—for instance, we also experimented with an analogous system that uses features based on discovered structure to guide a collaborative recommendation system [8].

5 Discovering complex structure for indirect use

Using structure indirectly (rather than directly) avoids many of the problems associated with discovering (rather than learning) structure. Meaningless structures can be filtered out by the statistical learning system—which is, after all, designed to handle irrelevant and noisy features. Importantly, structures can be used even if their precise semantics are unknown—meaningful structures can be exploited whenever they are correlated with the class to be predicted. In the settings described above, discovery algorithms that consider only a few types of structures are also less problematic; in principle, one need only run the discovery algorithm on more web pages to compensate for limited coverage.

For other applications, however, algorithms that can only discover simple structures from a limited class are much less useful. Consider the task of learning to identify executive biography pages (like the one shown in Figure 1) on company sites. One approach would be to train a classifier that uses as features the words in the page. Such a classifier might be trained by labeling pages from a number of representative company web sites, and then used to identify executive biography pages on new sites, such as the one of Figure 1.

This approach, however, ignores possibly useful structural information. For instance, on the web site for Figure 1, it might be that the executive biography pages are *exactly* those pages linked to by the anchors in the second column: in other words, the site might contain a structure that accurately identifies the target pages. Unfortunately, this structure cannot be used as a feature of the classifier, since it does not appear in the training data at all (recall the training data is taken from other web sites); it can only be used if it is somehow discovered "on the fly" when the pages on this site are encountered by the classifier.

Notice that in this setting, a algorithm that discovers only simple structures is of limited use: since only a few structures on a site will be informative, it is essential to be able to find all of them. This suggests adapting wrapper-learning machinery for finding complex structures to this task. In previous work [7], we proposed the following method for processing a web site.

First, all pages on a site are classified using a word-based classifier, and all anchors that point to an "executive biography" page are marked as positive. These markings are shown in Figure 1 with "'+" signs, and some errors are likely to be made. In the figure, the link to “Customers” is mistakenly marked positive, and the link to “Boris Badinov” is mistakenly unmarked.

Next, all sets of at most k nearby positive anchors are generated, for $k = 1, 2, 3$, and each such set is passed as input to each of the builders of WL^2. This produces a large number of structures D: some simple structures, produced from a single positive examples, and some more complex ones, produced from two or three nearby examples. Each discovered structure D is then used to generate new a feature f_D, which is true for all pages pointed to by some anchor in structure D. Finally, a new classifier is learned for this site, using the predictions provided by the original word-based classifier as labels, and using the new structure-based features to represent a page. The overall effect of this process is to smooth the predictions made by the word-based classifier toward sets of pages that are easily described by structures on the site. In our experiments, this smoothing process decreased the error rate of the original word-based page classifier by a factor of about half, on average.

In previous work, a number of other researchers have used hyperlink structure (hubs) to improve page classifiers [10, 17, 27] in a similar manner. Blei *et al* [1] also used the same structure-discovery algorithms in conjunction with a more complex probabilistic scheme for combining structural features with word-based predictions, and achieved a significant reduction in error rate on entity extraction tasks.

6 Conclusion

‘

Techniques that automatically learn and discover web-page structure are important for efforts to “understand” web pages. Techniques for learning web page structure are crucial for wrapper-learning, an important and active research area. They can also be beneficial for tasks as diverse: as classification of entities mentioned on the web; collaborative filtering for music; web page classification, and entity extraction from web pages.

References

[1] David M. Blei, J. Andrew Bagnell, and Andrew K. McCallum. Learning with scope, with application to information extraction and classification. In *Proceedings of UAI-2002*, Edmonton, Alberta, 2002.

[2] Boris Chidlovskii. Information extraction from tree documents by learning subtree delimiters. In *Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, August 2003. On line at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm.

[3] William Cohen, Andrew McCallum, and Dallan Quass. Learning to understand the web. *IEEE Data Engineering Bulletin*, 23:17–24, September 2000.

[4] William W. Cohen. Reasoning about textual similarity in information access. *Autonomous Agents and Multi-Agent Systems*, pages 65–86, 1999.

[5] William W. Cohen. Recognizing structure in web pages using similarity queries. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, FL, 1999.

[6] William W. Cohen. Automatically extracting features for concept learning from the web. In *Machine Learning: Proceedings of the Seventeeth International Conference*, Palo Alto, California, 2000. Morgan Kaufmann.

[7] William W. Cohen. Improving a page classifier with anchor extraction and link analysis. In *Advances in Neural Processing Systems 15*, Vancouver, British Columbia, 2002.

[8] William W. Cohen and Wei Fan. Web-collaborative filtering: Recommending music by crawling the web. In *Proceedings of The Ninth International World Wide Web Conference (WWW-2000)*, Amsterdam, 2000.

[9] William W. Cohen, Lee S. Jensen, and Matthew Hurst. A flexible learning system for wrapping tables and lists in HTML documents. In *Proceedings of The Eleventh International World Wide Web Conference (WWW-2002)*, Honolulu, Hawaii, 2002.

[10] David Cohn and Thomas Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.

[11] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[12] D.W. Embley, Y.S. Jiang, and W.-K. Ng. Record-boundary discovery in web documents. In *SIGMOD'99 Proceedings*, 1999.

[13] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from html tables with unknown structure. In *Proceedings of the 21st International Conference on Conceptual Modeling*, Tampere, Finland, October 2002.

[14] D. Freitag and N. Kushmeric. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, 2000.

[15] Chun-Nan Hsu, Chia-Hui Chang, Harianto Siek, Jiann-Jyh Lu, and Jen-Jie Chiou. Reconfigurable web wrapper agents for web information integration. In *Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, August 2003. On line at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm.

[16] Lee S. Jensen and William W. Cohen. Grouping extracted fields. In *Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, WA, 2001.

[17] T. Joachims, N. Cristianini, and J. Shawe-Taylor. Composite kernels for hypertext categorisation. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, 2001.

[18] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[19] N. Kushmeric. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.

[20] Nicholas Kushmeric, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Osaka, Japan, 1997.

[21] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[22] Kristina Lerman, Craig A. Knoblock, and Steven Minton. Automatic data extraction from lists and tables in web sources. In *Proceedings of the Automatic Text Extraction and Mining Workshop (ATEM-01)*, Seattle, WA, August 2001. Held in conjunction with IJCAI-01.

[23] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, august 1996.

[24] A. K. McCallum, K. Nigam, J. Rennie, , and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000.

[25] Steven N. Minton, Sorinel I. Ticrea, and Jennifer Beach. Trainability: Developing a responsive learning system. In *Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, August 2003. On line at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm.

[26] Ion Muslea, Steven Minton, and Craig Knoblock. Wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(12), 1999.

[27] S. Slattery and T. Mitchell. Discovering test set regularities in relational domains. In *Proceedings of the 17th International Conference on Machine Learning (ICML-2000)*, June 2000.

Structure Discovery using Statistical Relational Learning

Lise Getoor
Computer Science Department & UMIACS
University of Maryland
College Park, MD 20912
getoor@cs.umd.edu

Abstract

Statistical relational learning is a newly emerging area of machine learning that combines statistical modeling with relational representations. Here we argue that it provides a unified framework for the discovery of structural information that can be exploited by a data management system. The categories of structure that can be discovered include: instance-level dependencies and correlations, for example intra-table column dependencies and inter-table join dependencies; record linkages and duplicates; and schema matching and schema discovery from unstructured and semi-structured data.

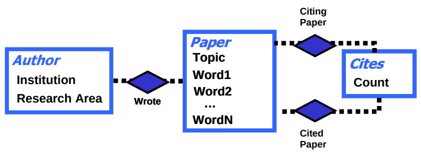
1 Introduction

Combining statistical approaches to machine learning with relational learning methods is receiving increasing attention. There have been several recent workshops held, including two on "Learning Statistical Models from Relational Data" [25, 26], two on "Multi-relational Data Mining" [18, 19] and a summer school on relational data mining [20].

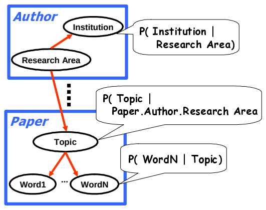
Statistical relational learning allows the combination of meta-information given by the domain schema and other background knowledge with principled approaches to model construction and inference based on statistical machine learning theory. A number of different models and formalisms have been proposed. One approach is probabilistic relational models (PRMs) [34]. Other approaches include Bayesian Logic Programs [33], Stochastic Logic Programs [42] and others. Here, we give a brief introduction to PRMs (for more details see [23]). First, we focus on their application to the discovery of instance-level structure. Next, we propose ways that they can be used to discover additional meta-level structure such as mappings and transformations. A key advantage is the ability to have a unified probabilistic framework for reasoning about data and meta-data together.

2 Probabilistic Relational Models

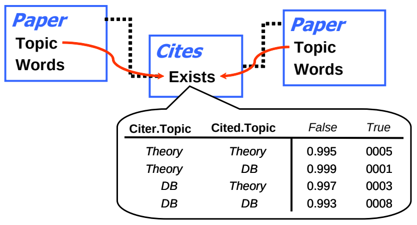
Probabilistic relational models (PRMs) are a recent development [34, 48, 53] that extend the standard attribute-based Bayesian network representation to incorporate a much richer relational structure. These models allow the specification of a probability model for *classes* of objects rather than for simple attributes; they also allow properties of an entity to depend probabilistically on properties of other *related* entities. The probabilistic class



(a)


Author
Institution
Research Area
P(Institution | Research Area)
Paper
Topic
P(Topic | Paper.Author.Research Area
Word1
WordN
P(WordN | Topic)

(b)



(c)

Figure 1: (a) The relational schema for a simple bibliographic domain. (b) The probabilistic dependencies for a simple bibliographic domain. (c) A probabilistic model for the existence of a citation.

model represents a generic dependence, which is then instantiated for specific circumstances, i.e., for particular sets of entities and relations between them.

As in the case of Bayesian networks, a key advantage of the models is that they provide a compact representation for a joint distribution over a set of random variables. The key that allows this compression is the discovery of independencies, and more importantly, conditional independencies, that hold in the joint distribution. Conditional independence is structure that can be discovered in the data; once discovered, it gives guidance in factoring the distribution. The collection of conditional independencies are constraints (or approximate constraints) that we can discover and exploit.

One component of a PRM is a relational schema for the domain, describing the tables, the attributes (columns) in the tables and the possible foreign-key joins between tables. Figure 1(a) shows a simplified relational schema for a bibliographic domain. The relational schema describes the categories of objects in the domain (here we have Papers, Authors, and Citations), the attributes of the objects (words, topic, count, institution and research area) and the foreign-key joins between the objects (written-by, citing paper and cited paper).

In addition to the relational component, the PRM describes the probabilistic relationships between attributes. Figure 1(b) shows a portion of the probabilistic dependency model. The direct probabilistic dependencies are indicated by arrows in the figure. Attributes are potentially correlated when there is a path between them. In addition, the probabilistic component also describes how the attributes interact by providing a local probability model or conditional probability distribution (CPD) for an attribute given its parents. These are indicated by the callouts in the figure. The local probability model describes the distribution of the child attribute given different values of the parents. In the case where the attributes are categorical, the local probability model is commonly a multinomial, or a tree-structured representation of the conditional distribution [8].

Importantly, PRMs can also model the probability of the existence of links between entities. In [24], we describe several approaches. One of these is to directly model the probability of the existence of a link. However, rather than build a model that uses the identity or key of the individual objects, which would be both unwieldy and would not support generalization, we base our probability model on attributes of the objects. Figure 1(c) shows a simple model for the existence of a citation between two papers. In this case the probability is based on the topics of the papers; papers that are on the same topic are more likely to cite each other.

A relational skeleton σ specifies the objects in our distribution. It defines the set of random variables of interest. A PRM describes the dependency structure $\mathcal{S}$ between the attributes, and the parameters of the CPDs in the model, $\theta_{\mathcal{S}}$. The semantics of a PRM specifies a probability distribution over the possible joint assignments $\mathcal{I}$ to the random variables of σ. As with Bayesian networks, the joint distribution over these assignments can be factored. That is, we take the product, over all $x.A$, of the probability in the CPD of the specific value assigned by the instance to the attribute given the values assigned to its parents. Formally, this is written as follows:

$$P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) \;=\; \prod_{X_i} \prod_{A \in \mathcal{A}(X_i)} \prod_{x \in \sigma(X_i)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\text{Pa}(x.A)}) \tag{1}$$

where $\mathcal{I}_{x.A}$, and $\mathcal{I}_{\text{Pa}(x.A)}$ are the values of a particular attribute and the values of its parent in $\mathcal{I}$. This expression is very similar to the chain rule for Bayesian networks. There are two primary differences. First, our random variables are the attributes of a set of objects. Second, the set of parents of a random variable can vary according to the relational context of the object — the set of objects to which it is related.

Learning Probabilistic Relational Models In order to learn a PRM from an existing database, we adapt and extend the machinery that has been developed over the years for learning Bayesian networks from data [30, 9, 35, 13] to the task of learning PRMs from structured data [21, 23]. In the learning problem, our input contains a relational schema that specifies the basic vocabulary in the domain — the set of tables, the attributes of each table and the possible foreign-key joins between tuples in the different tables. Our training data consists of a fully specified instance of that schema stored in a database.

There are two components of the learning task: parameter estimation and structure learning. In the parameter estimation task, we assume that the qualitative dependency structure of the PRM is known; i.e., the input consists of the schema and training database (as above), as well as a qualitative dependency structure $\mathcal{S}$. The learning task is only to fill in the parameters that define the CPDs of the attributes. In the structure learning task, we must discover the dependency structure $\mathcal{S}$ as well.

Parameter Estimation We are given the structure $\mathcal{S}$ that determines the set of parents for each attribute, and our task is to learn the parameters $\theta_{\mathcal{S}}$ that define the CPDs for this structure. The key ingredient in parameter estimation is the *likelihood function*, the probability of the data given the model. This function measures the extent to which the parameters provide a good explanation of the data. Intuitively, the higher the probability of the data given the model, the better the ability of the model to predict the data. The likelihood of a parameter set is defined to be the probability of the data given the model: $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) = P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}})$. We can perform *maximum likelihood* parameter estimation or we can take a *Bayesian approach* to parameter estimation by incorporating parameter priors. For an appropriate form of the prior and by making standard assumptions, we can get a closed form solution for either of these estimates. The key computation required is the calculation of the sufficient statistics, or counts, for each CPDs in the PRM. The straightforward implementation for this requires a pass over the database; intelligent use of previously computed statistics can have significant impact on efficiency when more than one model must be evaluated.

Structure Learning Next we are faced with the more challenging problem of learning a dependency structure automatically. The main problem here is finding a good dependency structure among the huge number of many possible ones. As in most learning algorithms, there are three important components that need to be defined: the **hypothesis space** which specifies which structures are candidate hypotheses that our learning algorithm can return; a **scoring function** that evaluates the "goodness" of different candidate hypotheses relative to the data; and the **search algorithm**, a procedure that searches the hypothesis space for a structure with a high score. We use a greedy local search; see [21, 22] for more detail. This search is heuristic; the algorithm is not guaranteed to find a global optima. The computational complexity of the algorithm depends of the size of the hypothesis space and the score landscape. In practice, we have found for medium-size data sets (that fit in main memory), the running time for the learning algorithms are quite reasonable, ranging from a few minutes to a half hour.

Making use of a Learned Probabilistic Relational Model A learned PRM provides a statistical model that can uncover and discover many interesting probabilistic dependencies that hold in a domain. Unlike a set of (probabilistic) rules for classification, PRMs specify a joint distribution over a relational domain. Thus, like Bayesian networks, they can be used for answering queries about any aspect of the domain given any set of observations. Furthermore, rather than trying to predict one particular attribute, the PRM learning algorithm attempts to tease out the most significant direct dependencies in the data. The resulting model thus provides a high-level, qualitative picture of the structure of the domain, in addition to the quantitative information provided by the probability distribution. Thus, PRMs are ideally suited to exploratory analysis of a domain and relational data mining.

We have applied these techniques with great success in a variety of domains. One of the interesting domains has been a study of tuberculosis patients, along with their contacts and genetic information about the TB strains [28]. Another very different application of these methods is to the classic problem of selectivity estimation in databases. We have shown that these structured relational models can be constructed and used to give very accurate estimates for a wide class of queries [29].

3 Identity Uncertainty

The above describes how probabilistic models can be used to discover structure in the data; both structure in the distribution of attribute values, and structure in the existence of links, or joins, between elements. Another structure that probabilistic relational models can discover is instance-level dependencies, for example when two records refer to the same individual. In many practical problems, such as information extraction, duplicate elimination and citation matching, objects may not have unique identifiers. The challenge is to determine when two similar-looking items in fact refer to the same object. This problem has been studied in statistics under the umbrella of record linkage [61, 62]; in the database community for the task of duplicate elimination [57]; and in the artificial intelligence community as identity uncertainty [56, 50, 5].

In the statistical relational learning setting, it is important to take into account not just the similarity of objects based on their attributes, but also based on their links. In a bibliographic setting, this means taking into account the citations of a paper. Note that as matches are identified, new matches may become apparent. Pasula et al. [50, 39] have studied the use of probabilistic relational models for citation matching. They are able to explicitly reason about the individuals in the domain, and whether two citations actually refer to the same individual. This becomes a complex inference task, but they are able to solve it using Markov chain Monte Carlo techniques.

4 Schema Matching and Mapping

Interest in applying machine learning to schema matching has been growing [17, 16]. Statistical relational learning can also be used for schema mapping and discovery. By making full use of the data distribution, the meta information, we can perform more accurate mappings. Rahm et al. [55] give a taxonomy of schema matching approaches. Probabilistic approaches can provide the foundation on which to combine element and structure-level schema matching with element and instance-based data content matching. Because they consider instance-level semantics, probablistic approaches also show promise in developing not just syntactic matchings between schemas, but mappings with a well-founded data semantics such as those used in the Clio system [40].

5 Schema Discovery

Applications of statistical relational learning to semi-structured and unstructured data range from information extraction from unstructured text to complex wrapper generation [12]; however the majority of these approaches assume that some schema information is given.

Automatic schema extraction from lists and tables is ideally suited to the use of probabilistic models. Probabilistic models can reason over different segmentations of the data, and can find the appropriate number of elements. Lerman et al. [37] describe a heuristic approach that automatically learns a schema using information from list and detail pages on web sites. A probabilistic approach can also be taken. The probabilistic method has the advantage that it can more easily handle noise (missing fields) and it can combine the evidence from multiple sources (i.e. the list and detail pages) in a principled manner.

6 Statistical Modeling Challenges

Statistical machine learning provides the theory for parameter estimation, model selection, prediction and inference. Regardless of the particular model family, there are some unique challenges to applying statistical modeling techniques to structure discovery in multi-relational domains.

Logical versus Statistical Dependencies: The first challenge in structure discovery is coherently handling two different types of dependence structures:

- **relational or link structure -** the logical relationships between objects
- **probabilistic dependency -** the statistical relationship between attributes of objects.

In learning statistical models for relational data, we must not only search over probabilistic dependencies, as is standard in any type of statistical model selection problem, but potentially we must search over the different possible logical relationships between objects. This search over logical relationships has been a focus of research in inductive logic programming [41, 36] and some of the methods and machinery developed in this community may be applicable here. Methods for inferring functional dependencies and multi-valued dependencies [32, 2, 1, 3] are also important. The search over probabilistic dependencies has been studied, for example in structure learning for Bayesian networks [13, 30] and in the work described earlier on learning probabilistic relational models. There is an opportunity to integrate these more tightly.

Feature Construction: A second challenge is feature construction in the relational setting [14, 52, 51]. The attributes of an object provide a basic description of the object. Traditional classification algorithms are based on these types of object features. In a relational approach, it may also make sense to use attributes of linked objects. Further, if the links themselves have attributes, these may also be used. However, as others have noted, simply flattening the relational neighborhood around an object can be problematic. For example in hypertext domains, simply including words from neighboring pages degrades classification performance [10, 49]. A further issue is how to deal appropriately with relationships that are not one-to-one. In this case, it may be appropriate to compute *aggregate* features over the set of related objects. We have shown that this works well for learning probabilistic relational models [23]; others have examined it in the context of ILP approaches [51]. In [38], we show that some simple link statistics computed based on an object's links can improve predictive accuracy. The feature construction in relational domains can be used by a range of statistical models, including the models described above, first-order decision trees [7, 47], relational neural networks [6] and others.

Collective Classification: A third challenge is statistical inference using a learned statistical relational model. A learned model specifies a distribution over link and content attributes, which may be correlated based on the links between them. Intuitively, for linked objects, making a prediction for the category of one object can influence the inferred categories of its linked neighbors. This requires a more complex classification algorithm than for a traditional non-relational statistical model. The predictions that need to be made are all correlated. This is one of the fundamental reasons why one cannot simply convert a relational learning problem into a collection of independent propositional inference tasks.

This complicates things, but at the same time allows for more accurate inferences to be made. Iterative classification algorithms have been proposed for hypertext categorization [10, 49] and for relational learning

[46, 59, 58]. The general approach of iterative classification has been studied in numerous fields, including relaxation-labeling in computer vision [31], inference in Markov random fields [11] and loopy belief propagation in Bayesian networks [44]. Some approaches make assumptions about the influence of the neighbor's categories (such as that linked objects have similar categories); we believe it is important to *learn* how the link distribution affects the category. Other approaches to collective classification assume a very regular logical structure, for example either a chain, such as HMMs [54] or a dynamic Bayesian network [43], or a grid. Approaches that can handle more general relational structures are needed.

Instances versus Classes: A final modeling challenge is whether our statistical models refer explicitly to individuals, or only to classes or categories of objects [27, 15]. In many cases, we'd like to model that a connection to a particular object or individual is highly predictive; on the other hand, if we'd like to have our models generalize and be applicable to new, unseen objects, we also have to be able to model with and reason about generic collections of objects. The need to reason explicitly about individuals is a fundamental element of link-based object identification. But it also ties back to the feature construction problem; it may be useful to have features that refer to particular individuals. The majority of current approaches either only model classes of objects [21], or only reason about individuals [60, 4]. We have some initial results in the context of probabilistic relational models [22]. This is a general issue that is not specific to any particular model choice.

Computational Complexity and Scaling: The scale and dimensionality of statistical models that can be tackled has increased by several orders of magnitude in recent years. Models that were considered too complex for analysis just a decade ago are now routinely solved using large scale Markov chain Monte Carlo simulations [45].

While there have been great strides made in the size and complexity of models now amenable to computational analysis, the size of data bases and corpora has grown even faster. There is gap, and it is not clear that whether the gap is getting wider or narrowing. Regardless, it is our opinion that the use of coherent complete probabilistic frameworks hold greater promise for scaling than collections of independent ad-hoc procedures. Even in cases where the complete framework proves intractable, analysis of the complete framework will give guidance for the best places to make approximations, and will give tools to do analysis and comparison to see the effect of the approximations.

7 Conclusion

We have given a brief introduction to statistical relational learning, and have argued that it is ideally suited to data source structure discovery. Probabilistic models that support both data-level and meta-level representation and reasoning provide a unified framework for representing instance-level structure, object-level structure (for record consolidation) and schema- level structure (for schema mapping and schema discovery).

References

[1] S. Bell. Discovery and maintenance of functional dependencies by independencies. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 27–32. AAAI Press, 1995.

[2] S. Bell. Dependency mining in relational databases. In Dov M. Gabbay, Rudolf Kruse, Andreas Nonnengart, and Hans Jürgen Ohlbach, editors, *Proceedings of the First International Joint Conference on Qualitative and Quantitative Practical Reasoning*, volume 1244 of *LNAI*, pages 16–29, Berlin, June9–12 1997. Springer.

[3] S. Bell and P. Brockhausen. Discovery of constraints and data dependencies in relational databases. In Nada Lavrač and Stefan Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *LNAI*, pages 267–270, Berlin, April 1995. Springer.

[4] A. Bernstein, S. Clearwater, and F. Provost. The relational vector-space model and industry classification. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 8–18, 2003.

[5] M. Bilenko and R. Mooney. On evaluation and training-set construction for duplicate detection. In *Proceedings of Knowledge Discovery and Data Mining*, pages 39–48, 2003.

[6] H. Blockeel and M. Bruynooghe. Aggregation versus selection bias, and relational neural networks. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 22–23, 2003.

[7] H. Blockeel and L. DeRaedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[8] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 115–123. Morgan Kaufman, August 1996.

[9] W. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[10] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 307–318, Seattle, Washington, 1998.

[11] R. Chellappa and A.K. Jain. *Markov random fields: theory and applications*. Academic Press, Boston, 1993.

[12] W. Cohen. Learning and discovering structure in web pages. *Data Engineering Bulletin*, 2003. in this volume.

[13] G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[14] C. Cumby and D. Roth. Feature extraction languages for propositionalized relational learning. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 24–31, 2003.

[15] J. Cussens. Individuals, relations and structures in probabilistic models. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 32–26, 2003.

[16] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 509–520, 2001.

[17] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: a multistrategy approach. *Machine Learning Journal*, 50(3):279–301, 2003.

[18] S. Dzeroski, L. De Raedt, and S. Wrobel. *Proc. KDD-2002 Workshop on Multi-Relational Data Mining*. ACM Press, 2002.

[19] S. Dzeroski, L. De Raedt, and S. Wrobel. *Proc. KDD-2003 Workshop on Multi-Relational Data Mining*. ACM Press, 2003.

[20] S. Dzeroski and B. Zenko. A report on the summer school on relational data mining. *SIGKDD Explorations*, 5:100–102, 2003.

[21] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1300–1307, Stockholm, Sweden, 1999. Morgan Kaufman.

[22] L. Getoor. *Learning Statistical Models of Relational Data*. PhD thesis, Stanford University, 2001.

[23] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 307–335. Kluwer, 2001.

[24] L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679–707, 2002.

[25] L. Getoor and D. Jensen. *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*. AAAI Press, 2000.

[26] L. Getoor and D. Jensen. *Proc. IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*. AAAI Press, 2003.

[27] L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries*, pages 25–34, 2000.

[28] L. Getoor, J. Rhee, D. Koller, and P. Small. Understanding tuberculosis epidemiology using probabilistic relational models. *Artificial Intelligence in Medicine Journal*, 2003. to appear.

[29] L. Getoor, B. Taskar, and D. Koller. Using probabilistic models for selectivity estimation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 461–472. ACM Press, 2001.

[30] D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 301–354. MIT Press, Cambridge, MA, 1998.

[31] R. Hummel and S. Zucker. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(5):267–287, 1983.

[32] M. Kantola, H. Mannila, K.-J. Rih, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *International Journal of Intelligent Systems*, 7(7):591–607, September 1992.

[33] K. Kersting, L. de Raedt, and S. Kramer. Interpreting Bayesian logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 29–35. AAAI Press, 2000.

[34] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 580–587, Madison, WI, 1998. AAAI Press.

[35] W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

[36] N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, NY, 1994.

[37] K. Lerman, C. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *IJCAI Workshop on Automatic Text Extraction and Mining*, 2001.

[38] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the Twentieth International Conference on Machine Learning*, Washington, DC, 2003.

[39] B. Marthi, B. Milch, and Stuart Russell. First-order probabilistic models for information extraction. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, 2003.

[40] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 77–88, Cairo, Egypt, September 2000.

[41] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, London, UK, 1992.

[42] S.H. Muggleton. Learning stochastic logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 36–41. AAAI Press, 2000.

[43] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Computer Science Dept, UC Berkeley, 2002.

[44] K. Murphy and Y. Weiss. Loopy belief propagation for approximate inference: an empirical study. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufman, 1999.

[45] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[46] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20. AAAI Press, 2000.

[47] J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. Technical Report 02-55, University of Massachusetts Amherst, 2002.

[48] L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1996.

[49] H. Oh, S. Myaeng, and M. Lee. A practical hypertext categorization method using links and incrementally available class information. In *Proc. of SIGIR-00*, pages 264–271, 2000.

[50] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*. MIT Press, 2003.

[51] C. Perlich and F. Provost. Aggregation-based feature invention and relational concept classes. In *Proceedings of Knowledge Discovery and Data Mining*, pages 167–176, 2003.

[52] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[53] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.

[54] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[55] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[56] S. Russell. Identity uncertainty. In *Proc. of IFSA-01*, Vancouver, 2001.

[57] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pages 269–278, 2002.

[58] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. of UAI-02*, pages 485–492, Edmonton, Canada, 2002.

[59] B. Taskar, E. Segal, and D. Koller. Probabilistic classification and clustering in relational data. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 870–876, Seattle, Washington, 2001. Morgan Kaufman.

[60] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[61] W. E. Winkler. Advanced methods for record linkage. Technical report, Statistical Research Division, U.S. Census Bureau, 1994.

[62] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Statistical Research Division, U.S. Census Bureau, 1994.

DTD Inference from XML Documents: The XTRACT Approach

Minos Garofalakis
Bell Laboratories
minos@bell-labs.com

Aristides Gionis[*]
University of Helsinki
gionis@cs.helsinki.fi

Rajeev Rastogi
Bell Laboratories
rastogi@bell-labs.com

S. Seshadri[*]
Strand Genomics
seshadri@strandgenomics.com

Kyuseok Shim
SNU and AITrc[†]
shim@ee.snu.ac.kr

Abstract

XML is rapidly emerging as the new standard for data representation and exchange on the Web. Document Type Descriptors (DTDs) contain valuable information on the structure of XML documents and thus have a crucial role in the efficient storage and querying of XML data. Despite their importance, however, DTDs are not mandatory, and it is quite possible for documents in XML databases to not have accompanying DTDs. In this paper, we present an overview of XTRACT, a novel system for inferring a DTD schema for a database of XML documents. Since the DTD syntax incorporates the full expressive power of regular expressions, naive approaches typically fail to produce concise and intuitive DTDs. Instead, the XTRACT inference algorithms employ a sequence of sophisticated steps that involve: (1) finding patterns in the input sequences and replacing them with regular expressions to generate "general" candidate DTDs, (2) factoring candidate DTDs using adaptations of algorithms from the logic optimization literature, and (3) applying the Minimum Description Length (MDL) principle to find the best DTD among the candidates.

1 Introduction

The genesis of the eXtensible Markup Language (XML) was based on the thesis that structured documents can be freely exchanged and manipulated, if published in a standard, open format. Indeed, as a corroboration of the thesis, XML today promises to enable a suite of next-generation web applications ranging from intelligent web searching to electronic commerce. XML documents comprise hierarchically nested collections of *elements*, where each element can be either atomic (i.e., raw character data) or composite (i.e., a sequence of nested subelements). Further, *tags* stored with elements in an XML document describe the semantics of the data. Thus, XML data is hierarchically structured and self-describing.

A *Document Type Descriptor (DTD)* may optionally accompany an XML document and essentially serves the role of a schema specifying the internal structure of the document. Briefly , a DTD specifies for every

[*]This work was done while the author was with Bell Laboratories.
[†]Seoul National University and Advanced Information Technology Research Center.

element, the *regular expression* pattern that subelement sequences of the element need to conform to. In addition to enabling the free exchange of electronic documents through industry-wide standards, DTDs also provide the basic mechanism for defining the structure of the underlying XML data. As a consequence, DTDs play a crucial role in the efficient storage of XML data as well as the formulation, optimization, and processing of queries over a collection of XML documents [3, 4, 7]. Despite their importance, however, DTDs are *not mandatory* and an XML document may not always have an accompanying DTD. This is typically the case, for instance, when large volumes of XML documents are automatically generated from data stored in relational databases, flat files (e.g., HTML pages, bibliography files), or other semistructured data repositories.

Based on the above discussion on the virtues of a DTD, it is important to devise algorithms and tools that can infer an accurate, meaningful DTD for a given collection of XML documents (i.e., *instances* of the DTD). This is *not* an easy task. In contrast to simple structural descriptions or typings (e.g., [7, 11]), the DTD syntax incorporates the full specification power of regular expressions; thus, manually deducing such a DTD schema for even a small set of XML documents created by a user could prove to be a process of daunting complexity. Furthermore, naive approaches typically fail to deliver meaningful and intuitive DTD descriptions of the underlying data. Both problems are, of course, exacerbated for *large* XML document collections.

In this paper, we provide an overview of the architecture of XTRACT [5, 6], a novel system for inferring an accurate, meaningful DTD schema for a repository of XML documents. A naive and straightforward solution to our DTD-extraction problem would be to infer as the DTD for an element, a "concise" expression which describes *exactly* all the sequences of subelements nested within the element in the entire document collection. However, DTDs generated by this approach tend to be voluminous and unintuitive. In fact, we discover that accurate and meaningful DTD schemas that are also intuitive and appealing to humans tend to *generalize*. That is, "good" DTDs are typically regular expressions describing subelement sequences that *may not actually occur* in the input XML documents. (Note that this, in fact, is always the case for DTD regular expressions that correspond to infinite regular languages, e.g., DTDs containing one or more Kleene stars (*) [9].) In practice, however, there are numerous such candidate DTDs that generalize the subelement sequences in the input, and choosing the DTD that best describes the structure of these sequences is a non-trivial task. The XTRACT inference algorithms employ the following novel combination of sophisticated techniques to generate DTD schemas that effectively capture the structure of the input sequences.

- **Generalization.** As a first step, XTRACT employs novel heuristic algorithms for finding patterns in each input sequence and replacing them with appropriate regular expressions to produce more general candidate DTDs. The main goal of the generalization step is to judiciously introduce metacharacters (like Kleene stars) to produce regular subexpressions that generalize the patterns observed in the input sequences. Our generalization heuristics are based on the discovery of frequent, neighboring occurrences of subsequences and symbols within each input sequence. To avoid an explosion in the number of resulting patterns, our techniques are inspired by practical, real-life DTD examples.
- **Factoring.** As a second step, XTRACT *factors* common subexpressions from the generalized candidate DTDs obtained in the generalization step, in order to make them more concise. The factoring algorithms applied are appropriate adaptations of techniques from the logic optimization literature [1, 14].
- **Minimum Description Length (MDL) Principle.** In the final and most important step, XTRACT employs Rissanen's *Minimum Description Length* (MDL) principle [13] to derive an elegant mechanism for composing a near-optimal DTD schema from the set of candidate DTDs generated by the earlier two steps. (Our MDL-based notion of optimality is described later in the paper.) The MDL principle has its roots in information theory and, essentially, provides a principled, scientific definition of the optimal "theory/model" that can be inferred from a set of data examples [12]. Using MDL allows XTRACT to control the amount of generalization introduced in the inferred DTD in a principled and, at the same time, intuitively-appealing fashion. We demonstrate that selecting the optimal DTD based on the MDL principle has a direct and natural mapping to the *Facility Location Problem (FLP)*, which is known to be

```
<article>
  <title> A Relational Model for
       Large Shared Data Banks </title>
  <author>
    <name> E. F. Codd </name>
    <affil> IBM Research </affil>
  </author>
</article>
```

(a)

```
<!ELEMENT article(title, author*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author(name, affil)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT affil (#PCDATA)>
```

(b)

Figure 1: (a) An Example XML Document. (b) An Example DTD.

$\mathcal{NP}$-complete [8]. Fortunately, efficient approximation algorithms with guaranteed performance bounds have been proposed for the FLP in the literature [2], thus allowing us to efficiently compose the final DTD in a near-optimal manner.

We have implemented our XTRACT DTD-derivation algorithms and conducted an extensive experimental study with both real-life and synthetic DTDs that demonstrates the effectiveness of our approach. Due to space constraints, the discussion of our experimental findings as well as most of the details of the XTRACT algorithms have been omitted from this overview paper; interested readers are referred to [5, 6].

2 Formulating the DTD-Inference Problem

Quick Overview of XML and DTDs. An XML document, like an HTML document, consists of nested element structures starting with a root element. Subelements of an element can either be elements or simply character data. Figure 1(a) illustrates an example XML document, in which the root element (`article`) has two nested subelements (`title` and `author`), and the `author` element in turn has two nested subelements. The `title` element contains character data denoting the title of the article while the `name` element contains the name of the author of the article. The ordering of subelements within an element is significant in XML. Elements can also have zero or more attribute/value pairs that are stored within the element's start tag.

A DTD is a grammar for describing the structure of an XML document. A DTD constrains the structure of an element by specifying a regular expression that its subelement sequences have to conform to. Figure 1(b) illustrates a DTD that the XML document in Figure 1(a) conforms to. The DTD declaration syntax uses commas for sequencing, $|$ for (exclusive) `OR`, parenthesis for grouping and the meta-characters $?, ^*, ^+$ to denote, respectively, zero or one, zero or more, and one or more occurrences of the preceding term. As a special case, the DTD corresponding to an element can be `ANY` which allows an arbitrary XML fragment to be nested within the element. We should point out that real-life DTDs can get fairly complex and can sometimes contain several regular expressions terms with multiple levels of nesting (e.g., $((ab)^*c)^*$). (For brevity, we denote XML elements by a single lower-case letter; we also omit explicit commas in element sequences and regular expressions.)

Problem Formulation. Our goal is to infer a DTD for a collection of XML documents. Thus, for each element that appears in the XML documents, we aim to derive a regular expression that subelement sequences for the element (in the XML documents) conform to. Note that an element's DTD is completely independent of the DTD for other elements, and only restricts the sequence of subelements nested within the element. Therefore, for simplicity of exposition, we concentrate on the problem of extracting a DTD for a single element.

Let e be an element that appears in the XML documents for which we want to infer the DTD. It is straightforward to compute the sequence of subelements nested within each $< e >< /e >$ pair in the XML documents. Let I denote the set of N such sequences, one sequence for every occurrence of element e in the data. The problem we address in this paper can be stated as follows: "*Given a set I of N input sequences nested within element e, compute a DTD for e such that every sequence in I conforms to the DTD.*"

As stated, an obvious solution to the problem is to find the most "concise" regular expression R whose language is I. One mechanism to find such a regular expression is to factor as much as possible, the expression corresponding to the `OR` of sequences in I. Factoring a regular expression makes it "concise" without changing the language of the expression. For example, $ab|ac$ can be factored into $a(b|c)$. An alternate method for computing the most concise regular expression is to first find the automaton with the smallest number of states that accepts I and then derive the regular expression from the automaton. Such concise regular expressions whose language is exactly I, are unfortunately not "good" DTDs, in the sense that they tend to be voluminous and unintuitive. We illustrate this using the DTD of Figure 1(b). Suppose we have a collection of XML documents that conform to this DTD. Abbreviating the `title` tag by t, and the `author` tag by a, it is reasonable to expect the following sequences to be the subelement sequences of the `article` element in the collection of XML documents: t, ta, taa, $taaa$, $taaaa$. Clearly, the most concise regular expression for the above language is $t|t(a|a(a|a(a|aa)))$ which is definitely much more voluminous and lot less intuitive than a DTD such as ta^*.

In other words, the obvious solution above never "generalizes" and would therefore never contain metacharacters like * in the inferred DTD. Clearly, a human being would at most times want to use such metacharacters in a DTD to succinctly convey the constraints he/she wishes to impose on the structure of XML documents. Thus, the challenge is to infer, for the set of input sequences I, a "general" DTD which is similar to what a human would come up with. However, as the following example illustrates, there can be several possible "generalizations" for a given set of input sequences and thus we need to devise a mechanism for choosing the one that best describes the sequences.

Example 1: Consider $I = \{ab, abab, ababab\}$. A number of DTDs match sequences in I – (1) $(a \mid b)^*$, (2) $ab \mid abab \mid ababab$, (3) $(ab)^*$, (4) $ab \mid ab(ab \mid abab)$, and so on. DTD (1) is similar to `ANY` in that it allows any arbitrary sequence of as and bs, while DTD (2) is simply an `OR` of all the sequences in I. DTD (4) is derived from DTD (2) by factoring the subsequence ab from the last two disjuncts of DTD (2). The problem with DTD (1) is that it represents a gross over-generalization of the input, and the inferred DTD completely fails to capture any structure inherent in the input. On the other hand, DTDs (2) and (4) accurately reflect the structure of the input sequences but do not generalize or learn any meaningful patterns which make the DTDs smaller or simpler to understand. Thus, none of the DTDs (1), (2), or (4) seem "good"; on the other hand, DTD (3) has great intuitive appeal since it is succinct and it generalizes the input sequences without losing too much information about their structure.

Based on the discussion in the above example, we can characterize the set of desirable DTDs by placing the following two qualitative restrictions on the inferred DTD: *(R1) The DTD should be concise (i.e., small in size);* and, *(R2) The DTD should be precise (i.e, not cover too many sequences not contained in I).* Restriction (R1) above ensures that the inferred DTD is easy to understand and succinct, thus eliminating, in many cases, exact solutions, i.e., regular expressions whose language is *exactly* I. Restriction (R2), on the other hand, attempts to ensure that the DTD is not too general and captures the structure of input sequences, thus eliminating trivial DTDs such as `ANY`. While the above restrictions seem reasonable at an intuitive level, there is a problem with devising a solution based on the above restrictions. The problem is that restrictions (R1) and (R2) conflict with each other. In our earlier example, restriction (R1) would favor DTDs (1) and (3), while these DTDs would not be considered good according to criterion (R2). The situation is exactly the reverse when we consider DTDs (2) and (4). Thus, in general, there is a tradeoff between a DTD's "conciseness" and its "preciseness", and a good DTD is one that strikes the right balance between the two. The problem here is that conciseness and preciseness are qualitative notions – in order to resolve the tradeoff between the two, we need to devise quantitative measures for mathematically capturing the two qualitative notions.

Using the MDL Principle to Define a Good DTD. We use the MDL principle [13] to define an information-theoretic measure for quantifying and thereby resolving the tradeoff between the conciseness and preciseness properties of DTDs. The MDL principle has been successfully applied in the past in a variety of situations ranging from constructing good decision tree classifiers [12] to learning common patterns in sets of strings [10].

Roughly speaking, the MDL principle states that the best theory to infer from a set of data is the one which minimizes the sum of: *(A) The length of the theory (in bits);* and, *(B) The length of the data (in bits), when encoded with the help of the theory*. We will refer to the above sum, for a theory, as the *MDL cost* for the theory. The MDL principle is a general one and needs to be instantiated appropriately for each situation. In our setting, the theory is the DTD and the data is the sequences in I. Thus, the MDL principle assigns each DTD an MDL cost and ranks the DTDs based on their MDL costs (DTDs with lower MDL costs are ranked higher). Furthermore, parts (A) and (B) of the MDL cost for a DTD depend directly on its conciseness and preciseness, respectively. Part (A) is the number of bits required to describe the DTD and is thus a direct measure of its conciseness. Further, since a DTD that is more precise captures the structure of the input sequences more accurately, fewer bits are required to describe the sequences in I in terms of a more precise DTD; as a result, part (B) of the MDL cost captures a DTD's preciseness. The MDL cost for a DTD thus provides us with an elegant and principled mechanism (rooted in information theory) for quantifying (and combining) the conflicting concepts of conciseness and preciseness in a single unified framework, and in a manner that is consistent with our intuition. By favoring concise and precise DTDs, and penalizing those that are not, it ranks highly exactly those DTDs that would be deemed desirable by humans.

Note that the actual encoding scheme used to specify a DTD as well as the data (in terms of the DTD) plays a critical role in determining the actual values for the two components of the MDL cost. The following example uses a simple, coarse encoding scheme to illustrate how ranking DTDs based on their MDL cost closely matches our intuition of their goodness; the details of XTRACT's encoding scheme can be found in [5, 6].

Example 2: Consider the input set I and DTDs from Example 1. We rank our example DTDs based on their MDL costs (DTDs with smaller MDL costs are considered better). (Our encoding assumes a cost of 1 unit for each character.) DTD (1), $(a \mid b)^*$, has a cost of 6 for encoding the DTD. In order to encode the sequence $abab$ using the DTD, we need one character to specify the number of repetitions of the term $(a \mid b)$ that precedes the * (in this case, this number is 4), and 4 additional characters to specify which of a or b is chosen from each repetition. Thus, the total cost of encoding $abab$ using $(a \mid b)^*$ is 5 and the MDL cost of the DTD is $6 + 3 + 5 + 7 = 21$. Similarly, the MDL cost of DTD (2) can be shown to be 14 (to encode the DTD) + 3 (to encode the input sequences; we need one character to specify the position of the disjunct for each sequence) = 17. The cost of DTD (3) is 5 (to encode the DTD) + 3 (to encode the input sequences – note that we only need to specify the number of repetitions of the term ab for each sequence) = 8. Finally, DTD (4) has a cost of 14 + 5 (1 character to encode sequence ab and 2 characters for each of the other two input sequences) = 19. Thus, DTD (3) is the best (i.e., lowest MDL cost) DTD in our example instance – which matches our intuition.

The above example shows that the MDL principle indeed provides an elegant mechanism for quantifying and resolving the tradeoff between the conciseness and preciseness properties of DTDs. More specifically: (1) the *"theory length"* part of the MDL cost includes the number of bits required to encode the DTD – this ensures that the inferred DTD is succinct; and, (2) the *"data length"* part of the MDL cost includes the number of bits needed for encoding the input sequences using the DTD – usually, expressing data in terms of a more general DTD (e.g., $(a \mid b)^*$ in Example 2) requires more bits than describing data in terms of a more specific DTD (e.g., $(ab)^*$ in Example 2). Thus, using the MDL principle enables us to choose a DTD that strikes the right balance between conciseness and preciseness.

3 Overview of the XTRACT System

The XTRACT system architecture is illustrated in Figure 2(a). XTRACT consists of three main components: the generalization module, the factoring module, and the MDL module. Input sequences in I are processed by the three subsystems one after another, the output of one subsystem serving as input to the next. We denote the outputs of the generalization and factoring modules by $\mathcal{S_G}$ and $\mathcal{S_F}$, respectively. Observe that both $\mathcal{S_G}$ and $\mathcal{S_F}$ contain the initial input sequences in I. This is to ensure that the MDL module has a wide range of DTDs

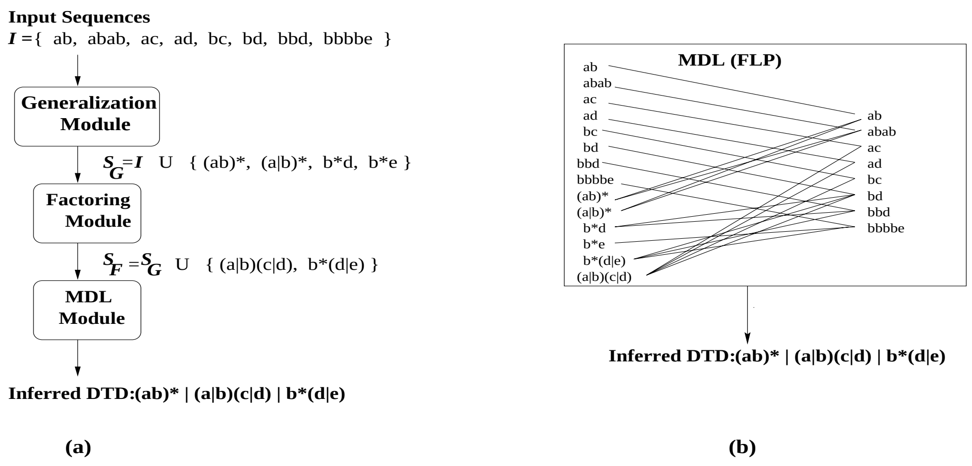

Input Sequences
I ={ ab, abab, ac, ad, bc, bd, bbd, bbbbe }
Generalization Module
S_G=I U { (ab)*, (a|b)*, b*d, b*e }
Factoring Module
S_F =S_G U { (a|b)(c|d), b*(d|e) }
MDL Module
Inferred DTD:(ab)* | (a|b)(c|d) | b*(d|e)
(a)
MDL (FLP)
ab
abab
ac
ad
bc
bd
bbd
bbbbe
(ab)*
(a|b)*
b*d
b*e
b*(d|e)
(a|b)(c|d)
ab
abab
ac
ad
bc
bd
bbd
bbbbe
Inferred DTD:(ab)* | (a|b)(c|d) | b*(d|e)
(b)

Figure 2: (a) XTRACT System Architecture. (b) XTRACT's MDL Subsystem.

to choose from that includes the obvious DTD which is simply an OR of all the input sequences in I. In the following, we provide a brief description of each XTRACT subsystem; more details can be found in [5, 6].

The Generalization Subsystem. For each input sequence, the generalization module generates zero or more candidate DTDs that are derived by replacing patterns in the input sequence with regular expressions containing metacharacters like * and | (e.g., $(ab)^*$, $(a \mid b)^*$). Note that the initial input sequences do not contain metacharacters and so the candidate DTDs introduced by the generalization module are more general. For instance, in Figure 2(a), sequences $abab$ and $bbbe$ result in the more general candidate DTDs $(ab)^*$, $(a \mid b)^*$ and b^*e to be output by the generalization subsystem. Also, observe that each candidate DTD produced by the generalization module may cover only a subset of the input sequences. Thus, the final DTD output by the MDL module may be an OR of multiple candidate DTDs.

Ideally, in the generalization phase, we should consider all DTDs that cover one or more input sequences as candidates so that the MDL step can choose the best among them. However, the number of such DTDs can be enormous. For example, the sequence $ababaabb$ is covered by the following DTDs in addition to many more – $(a \mid b)^*, (a \mid b)^*a^*b^*, (ab)^*(a \mid b)^*, (ab)^*a^*b^*$. Therefore, XTRACT employs several novel heuristics, inspired by real-life DTDs, for limiting the set of candidate DTDs $\mathcal{S}_\mathcal{G}$ output by the generalization module.

The Factoring Subsystem. The factoring component factors two or more candidate DTDs in $\mathcal{S}_\mathcal{G}$ into a new candidate DTD. The length of the new DTD is smaller than the sum of the sizes of the DTDs factored. For example, in Figure 2(a), candidate DTDs b^*d and b^*e representing the expression $b^*d \mid b^*e$, when factored, result in the DTD $b^*(d \mid e)$; similarly, the candidates ac, ad, bc and bd are factored into $(a \mid b)(c \mid d)$ (the pre-factored expression is $ac \mid ad \mid bc \mid bd$). Although factoring leaves the semantics of candidate DTDs unchanged, it is nevertheless an important step. The reason being that factoring reduces the size of the DTD and thus the cost of encoding the DTD, without seriously impacting the cost of encoding input sequences using the DTD. Thus, since the DTD encoding cost is a component of the MDL cost for a DTD, factoring can result in certain DTDs being chosen by the MDL module that may not have been considered earlier. We appropriately modify factoring algorithms for boolean functions in the logic optimization area [1, 14] to meet our needs. However, even though every subset of candidate DTDs can, in principle, be factored, the number of these subsets can be large and only a few of them result in good factorizations. We propose novel heuristics to restrict our attention to subsets that can be factored effectively.

The MDL Subsystem. The MDL subsystem finally chooses from among the set of candidate DTDs $\mathcal{S}_\mathcal{F}$ generated by the previous two subsystems, a (sub)set of DTDs $\mathcal{S}$ such that the final DTD (which is the OR of the DTDs in $\mathcal{S}$): (1) covers all the input sequences in I, and (2) has the *minimum MDL cost*. For the input sequences in Figure 2(a), we illustrate (using solid lines) in Figure 2(b), the input sequences (in the right column) covered by the candidate DTDs in $\mathcal{S}_{\mathcal{F}}$ (in the left column).

The above MDL-cost minimization problem naturally maps to the *Facility Location Problem (FLP)* [2, 8], which can be stated as follows: Let C be a set of clients and J be a set of facilities such that each facility "serves" every client. There is a cost $c(j)$ of "choosing" a facility $j \in J$ and a cost $d(j, i)$ of serving client $i \in C$ by facility $j \in J$. The problem definition asks to choose a subset of facilities $F \subset J$ such that the sum of costs of the chosen facilities plus the sum of costs of serving every client by its closest chosen facility is minimized.

Our problem of inferring the minimum MDL cost DTD can be reduced to FLP by letting C be the set I of input sequences and J be the set of candidate DTDs in $\mathcal{S}_{\mathcal{F}}$. The cost $c(j)$ of choosing a facility j is the length of the corresponding candidate DTD, whereas the cost $d(j, i)$ of serving client i from facility j is the length of the encoding of the sequence corresponding to i using the DTD corresponding to facility j.

The FLP is $\mathcal{NP}$-hard; however, it can be reduced to *Set Cover* and then approximated within a logarithmic factor as shown in [8]. Our XTRACT implementation employs the randomized algorithm from [2], which approximates the FLP to within a constant factor if the distance function is a metric. Even though our distance function is not a metric, we have found the solutions produced by [2] for our problem setting to be very good in practice. The final DTD inferred by XTRACT for our example instance is shown at the bottom of Figure 2(b).

4 Conclusions

In this paper, we have presented an overview of the XTRACT system [5, 6] for inferring a DTD for a database of XML documents. The DTD-inference problem is complicated by the fact that DTD syntax incorporates the full expressive power of regular expressions. Naive approaches that do not "generalize" beyond the input element sequences fail to deduce concise and semantically meaningful DTDs. Instead, XTRACT applies sophisticated algorithms that combine generalization and factorization steps with Rissanen's MDL principle in order to compute a DTD schema that is more along the lines of what a human would infer.

References

[1] R. K. Brayton and C. McMullen. The decomposition and factorization of boolean expressions. In *Intl. Symp. on Circuits and Systems*, 1982.

[2] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *40th Annual Symp. on Foundations of Computer Science*, 1999.

[3] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with stored. In *ACM SIGMOD Intl. Conf. on Management of Data*, 1999.

[4] M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Intl. Conf. on Database Theory (ICDT)*, 1997.

[5] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: Learning Document Type Descriptors from XML Document Collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, January 2003.

[6] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *ACM SIGMOD Intl. Conf. on Management of Data*, 2000.

[7] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *23rd Intl. Conf. on Very Large Data Bases*, 1997.

[8] D. S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22:148–162, 1982.

[9] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automaton Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[10] P. Kilpeläinen, H. Mannila, and E. Ukkonen. MDL learning of unions of simple pattern languages from positive examples. In *2nd European Conf. on Computational Learning Theory, EuroCOLT*, 1995.

[11] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *ACM SIGMOD Intl. Conf. on Management of Data*, 1998.

[12] J. Ross Quinlan and Ronald L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[13] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[14] A. R. R. Wang. *Algorithms for Multi-level Logic Optimization*. PhD thesis, Univ. of California, Berkeley, 1989.

Discovering Structure in a Corpus of Schemas

Alon Y. Halevy
University of Washington
alon@cs.washington.edu

Jayant Madhavan
University of Washington
jayant@cs.washington.edu

Philip A. Bernstein
Microsoft Research
philbe@microsoft.com

Abstract

This paper describes a research program that exploits a large corpus of database schemas, possibly with associated data and meta-data, to build tools that facilitate the creation, querying and sharing of structured data. The key insight is that given a large corpus, we can discover patterns concerning how designers create structures for representing domains. Given these patterns, we can more easily map between disparate structures or propose structures that are appropriate for a given domain. We describe the first application of our approach to the problem of semi-automatic schema matching.

1 Introduction

Database and knowledge base systems offer their users powerful mechanisms for querying their data. However, such power comes with a significant upfront cost, namely, getting the data into structured form. A key component of the cost is the effort involved in designing a database schema, and for the users, the effort of learning the schema in order to be able to query it. It is instructive to compare the conceptual effort involved in dealing with structured data versus the effort in dealing with text documents [HED$^+$03]. In the latter case, authoring data is a matter of writing coherent text and keyword querying is simple, though providing much less expressive power and accuracy. The differences between the management of structured and unstructured data are exacerbated when people try to *share* data (e.g., on the web, within enterprises and in large scientific projects). Mediating between disparate databases requires understanding the semantic relationships between the data sets which usually involves a large cooperative effort.

The goal of the Revere Project [HED$^+$03] at the University of Washington is to facilitate the activities of authoring, querying and sharing of structured data, so they become palatable to non-expert users. The main components of the project are (1) the Mangrove System [MEH$^+$03], whose goal is to entice people to structure their data, (2) the Piazza System [HITM03, HIST03, HIM$^+$04, MH03] that allows large-scale sharing of structured data without any central mediation, and (3) a set of tools that facilitate the authoring of schemas and the creation of semantic mappings between disparate schemas for data sharing.

This paper focuses on the latter component of Revere, which takes a new approach to developing tools for aiding the management of structured data. The techniques are inspired by one of the main ideas in Information Retrieval (IR), namely the statistical analysis of large corpora of text. Here we examine what we can do with a large corpus of database schemas, possibly with associated data instances and meta-data.

We begin in Section 2 by describing the contents of such a corpus and what we aim to achieve with it. In Section 3 we describe its first application to the problem of matching between disparate schemas. In Section 4 we discuss other uses of corpus-based tools, and in Section 5 we describe the main challenges we face in pursuing this approach.

2 A Corpus of Schemas

In Information Retrieval (IR), large corpora of text are analyzed for specific patterns: co-occurrences of terms often indicate that the terms are related or synonymous; high frequency of a term in a particular document (compared to the rest of the corpus) indicates the term's importance to the document. More generally, IR tries to glean clues from patterns in people's use of natural language.

In our context, our goal is to use a corpus of schemas to find patterns in *how people build structures for data*. In particular, creating a database schema involves at least the following steps. In each of these we can search for common patterns:

- Selecting entities and relationships between them: typically done using a conceptual entity/relationship model.
- Naming: choose particular names for the entities, their properties and the relationships, and
- Structuring: take the conceptual model and translate it into a particular data model (e.g., relational tables, object or XML hierarchies). While this step is aided by tools, there is still significant manual work. Furthermore, in this step additional naming decisions are made (e.g., for table names, disambiguating property names).

As in IR, patterns in the above processes cannot be written, but can only be discovered from a large collection of database structures. Once we discover the patterns, we can build tools that assist designers in authoring new schemas, querying schemas, and mapping between disparate schemas. We now briefly discuss the contents of a corpus and the kinds of patterns we can mine from it.

2.1 Contents of a corpus

A corpus can include any kind of information related to structured data, and may give us cues as to the meaning of the data. In particular, it includes the following.

1. Various forms of structures: on the less expressive end, the corpus can include relational and object-oriented database schema or entity/relationship diagrams, XML DTDs or schemas, possibly with associated integrity constraints (e.g., functional dependencies). On the more expressive end, the corpus can include terminologies in a knowledge representation language (e.g., in OWL [DCvH$^+$02]).

2. Instance data: it can include the actual rows of tables (or representative rows) or XML documents. In fact it can include data sets that do not have a schema (e.g., certain file formats). Very often, elements in the schema of one model are instance data in another model. Hence, the distinction between schema and instance data is not clear cut.

3. Validated mappings: previously constructed mappings between different database structures provide significant clues as to how entities are represented differently in disparate models. Whenever we have such mappings, they should be included in the corpus.

4. Queries: queries (posed by users or applications) provide important information about how certain data is used. For example, when a database query performs a join over attributes of two different tables, that indicates that the columns are modeling the same domain (and this often is not evident from the schema, which only specifies the data type).

5. Other meta-data: there are many forms of meta-data that accompany database structures. They range from text descriptions of the meaning of different fields to statistics about table cardinalities or histograms on the set of possible values within a column.

It is important to emphasize that a corpus is *not* a logically coherent universal database. Rather, it is a collection of disparate uncoordinated structures. We expect that the structures of the corpus will be stored and accessed using tools for model management [Ber03], which provide a set of operators for manipulating models of data (as opposed to the data itself).

2.1.1 Statistics on the corpus

There is a plethora of possible analyses that can be performed on such a corpus in order to extract interesting patterns. Finding the most effective ones is a long term research challenge. Below we describe certain kinds of statistics that can be computed over the corpus. We classify them according to whether they apply to individual words or terms, partial structures, or elements of particular schemas.

Word and term statistics: these statistics are associated with individual words (in any language) and with noun or verb phrases. These statistics indicate how a word is used in different roles in structured data. For each of these statistics, we can maintain different versions, depending on whether we take into consideration word stemming, synonym tables, inter-language dictionaries, or any combination of these three. The statistics include:

1. Term usage: How frequently the term is used as a relation name, as an attribute name, or in data (as a percentage of all of its uses or as a percentage of structures in the corpus).

2. Co-occurring schema elements: For each of the different uses of a term, which relation names and attributes tend to appear with it? What tend to be the names of related tables and their attribute names? What tend to be the join predicates on pairs of tables? Are there clusters of attribute names that appear together? Are there mutually exclusive uses of attribute names? We can also learn from co-occurrence of terms in *composite* names of schema elements.

3. Similar names: For each of the uses of a term, which other words tend to be used with similar statistical characteristics?

Composite statistics: the same statistics can be applied to *partial structures*. Examples of partial structures are sets of data instances, relations with associated attribute names, a relation with some data (possibly with missing values). In fact, the works in [HC03, KN03] are attempts to learn from partial structures.

Clearly, we need to significantly limit the number of partial structures for which we keep statistics (e.g., use techniques for discovering partial structures that occur frequently e.g., [PG02]). Given statistics for certain partial structures, we can estimate the statistics for other related structures.

Statistics for schema elements: the same word, used in different structures, can have different meanings. Hence, we may want to characterize the specific usages of terms in structures, and relate them to usage of terms in other structures. For example, in [MBC$^+$03] we learn a classifier for every relation and attribute name in the corpus. Following [DDH01], we use meta-strategy learning. The training data used for learning is gleaned from the schema to which the element belongs and the training data of elements that have been mapped to it by a validated mapping in the corpus. Intuitively, the classifier is meant to recognize the particular usage of the term, even if it appears differently in another structure.

3 Using a Corpus for Schema Matching

We now describe the first application of our corpus-based approach, schema matching. Sharing data among multiple data sources and applications is a problem that arises time and again in large enterprises, B2B settings, coordination between government agencies, large-scale science projects, and on the World-Wide Web. While there are many architectures for sharing data (data warehousing, data integration systems [Len02, DHW01], message passing systems (e.g., [MQS03]), web services, and peer-data management systems [HIST03, KNO+02, BGK+02, NWQ+02, AK03, ACMH03, NOTZ03]), a key problem in all of them is the semantic heterogeneity between the structures (e.g., schemas) of data sources that were originally designed independently. Thus, to obtain meaningful interoperation, one needs a *semantic mapping* between the schemas. A semantic mapping is a set of expressions that specify how the data in one database corresponds to the data in another database [MBDH02]. While languages for specifying semantic mappings have been developed and are well understood (see [Len02, Hal01] for surveys), the creation of semantic mappings has become a key bottleneck as it is labor intensive and error-prone.

The goal of schema matching is to assist a human to relate two domain models. Complete automation of the process is unlikely, so the goal is to significantly increase the productivity of human experts. The matching problem is difficult because it requires understanding the underlying semantics of the schemas being matched. While a schema (with its instance data) provides many clues to its intended semantics, it does not suffice in order to relate it to a different schema.

The process of generating a semantic mapping has traditionally been divided into two phases. The first phase finds a *match* between the two schemas. The match result is a set of *correspondences* between elements in the two schemas, stating that these elements are *somehow* related. For example, a correspondence may state that buyer in one schema model corresponds to customer in another. The second phase builds on the correspondences by creating the mapping expressions. The mapping expressions, often expressed as queries or rules, enable translating data from one data source to another, or reformulating a query over one data source into a query on the other. A plethora of techniques have been proposed for schema matching: see [RB01] for a survey, and [NM02, DMDH02, DR02] for some work since then. Collectively, these techniques mirror the heuristics that a human designer may follow. For example, techniques have considered exploiting relationships between names of elements in the schemas, structural similarities between them, similarities in data values, and even correlations between values in different attributes. Several recent works on schema matching are based on *combining* multiple techniques in a principled fashion [MBR01, DDH01, DR02].

Corpus-based matching

Schema matching is often facilitated by a detailed knowledge about the domain in which the matching is being performed. However, creating a knowledge base is often hard, and furthermore, the result may be brittle in the sense it only helps on its domain of coverage, and only provides a *single* perspective on the domain. Our approach, corpus-based matching, complements a knowledge base by gleaning relevant knowledge from a large corpus of database schemas and previously validated mappings. There are two types of knowledge that we can glean from such a corpus. First, we can learn the different ways in which words (or terms) are used in database structures (i.e., as relation names, attribute names and data values). Second, the validated mappings show how variations in term usages correspond to each other in disparate structures.

Although such a corpus is not easy to construct, it is a very different kind of activity than building a detailed and comprehensive knowledge base. It does not require the careful ontological design as a knowledge base does, nor the careful control of its contents, thereby removing key bottlenecks present in the design of knowledge bases. The corpus offers *multiple perspectives* on modeling a particular domain, including different levels of coverage of the domain. Thereby, it is more likely to provide knowledge that is useful for matching two disparate schemas.

The LSD System [DDH01] investigated the benefit of learning from previously validated mappings. That worked considered the case where multiple data sources are mapped to a single *mediated schema*, on which users pose queries. LSD was provided with the mediated schema and a set of *training matches* for some data sources. LSD used these matches to learn models of the elements of the mediated schema. Since no single learning algorithm captures all the cues from the domain, LSD used a multi-strategy approach that combined the predictions of several learners. LSD was then asked to predict matches between the mediated schema and a set of test schemas. The experiments in [DDH01] showed that (1) it is possible to achieve high accuracy with multi-strategy learning, and (2) additional accuracy is obtained by considering domain constraints (i.e., a simple form of domain knowledge). Overall, LSD achieved matching accuracy of 75-90% on small to medium sized schemas of data sources on the Web. LSD was then extended to consider simple taxonomies of concepts in [DMDH02].

In recent work [MBC$^+$03], we investigate the benefit of a corpus of schemas and matches, and the ability to use such a corpus to predict mappings between a *pair* of schemas that have not been previously seen. Like in LSD, we learn models for elements in the corpus, using both the information available in the schema and validated matches that are provided in the corpus. Given two schemas, S_1 and S_2, we calculate for each element in them a *similarity vector* w.r.t. the corpus, i.e., how similar each element in S_i is to each element in the corpus. Very roughly speaking, if the similarity vectors of two elements $a_1 \in S_1$ and $a_2 \in S_2$ are similar to each other, then we predict that a_1 matches a_2. The results of our experiments show that (1) even with a modest corpus of 10 schemas we are able to achieve good accuracy, and (2) the correct matches found by using the corpus and those found by other previously known techniques overlap, but have significant differences. Hence, the use of the corpus is finding matches that would not have been predicted by other techniques.

4 Other Uses of the Corpus

While schema matching was the first application that motivated our corpus-based approach, we believe the corpus is a general tool that is applicable elsewhere[1] We now briefly describe some of these applications.

4.1 Creating and Querying Structured Data

As argued at the outset, one of the greatest impediments to using database and knowledge base technology is the conceptual difficulty of dealing with structured data. Hence, we are faced with the challenge of creating tools that facilitate the creation and querying of structured data.

One such tool would be a *schema design advisor*, which assists in the authoring of structured data, much in the spirit of an auto-complete feature. A user of such a tool creates a schema fragment and some data in a particular domain, and the tool then proposes extensions to the schema using the corpus. The user may choose a schema from the list and modify it further to fit the local context. Note that besides time savings, such a tool has other advantages, such as possibly resulting in better designs and helping users conform to certain standards, when these are applicable.

On the querying side, the corpus-based approach can facilitate the querying of unfamiliar data. Specifically, consider a tool that enables you to pose a query using *your own terminology* to any database. The tool would then use the corpus to propose reformulations of your query that are well formed w.r.t. the schemas of the database at hand. The tool may propose a few such queries (possibly with example answers), and let you choose among them or refine them.

[1]In [HM03] we argue that the corpus can form a basis for a new class of Knowledge Representation systems.

4.2 Web Search and Query Answering

Another class of applications concerns various information finding tasks on the Web (or intranet). The first application is *query answering* (e.g., [KEW01, RFQ+02]): a natural language query is posed to a web search interface, and rather than finding relevant pages, the search engine tries to find the answers to the query. A second, related application is *focused crawling* (e.g., [SBG+03]), where the search engine is given a particular topic and tries to find pages relevant to it by crawling from an initial set of pages. In both of these applications, the presence of additional domain knowledge has been shown or argued to be useful. However, the cost of constructing knowledge bases with such wide domain coverage is prohibitive. In contrast, a corpus-based approach can yield a more robust solution.

5 Challenges

Our initial work has revealed significant challenges to building corpus-based tools. We briefly describe some of these challenges that provide many exciting research opportunities.

Creating the corpus: The first challenge is, of course, creating a corpus of interest. Naturally, organizations will not be quick to freely share their schemas. Fortunately, there are many publicly available schemas that are already useful. Furthermore, our initial experiments have shown that significant advantages can be obtained even by learning from a relatively small number of schemas.

A more subtle issue is the *focus* of the corpus – how closely do the domains of the schemas in the corpus need to be related to the domain for which the corpus is being used. For example, if we are concerned with mapping between disparate schemas of purchase orders, should our corpus include only schemas in this domain? Will schemas in other domains help or detract from the effectiveness of the corpus-based methods? Ideally, we would like to be able to collect a large corpus of schemas without carefully controlling their domains, and devise methods that exploit only the relevant information from the corpus.

Granularity in the corpus: The corpus will include many disparate schemas. At one extreme, we can view *every* term in *every* schema as a separate 'concept' and try to learn patterns about its usage. However, such an approach will quickly get out of hand. Instead, we need to devise techniques that cluster elements in the corpus into larger concepts and learn patterns about these clusters.

Tuning the corpus: Whatever automatic methods we use to analyze the corpus, there is no doubt that manual tuning of the corpus can be useful. The tuning can be of several forms: removing useless schemas, adding particularly useful mappings between schemas in the corpus, helping in clustering terms in the corpus, etc. In addition, combining the corpus with a manually constructed domain model raises interesting challenges.

In conclusion, while the challenges for corpus-based tools are enormous, we believe the payoffs could be huge, and the results can profoundly impact how we create and use structured data.

Acknowledgments

Many colleagues have contributed to the ideas that led to corpus-based representation. We would like to thank Anhai Doan, Pedro Domingos, Oren Etzioni, Zack Ives, Pradeep Shenoy, and Igor Tatarinov. Funding was provided by an NSF CAREER/PECASE Grant IIS-9985114, ITR Grant IIS-0205635, a Sloan Fellowship, and gifts from Microsoft Research, Intel, NEC and NTT.

References

[ACMH03] Karl Aberer, Philippe Cudre-Mauroux, and Manfred Hauswirth. The chatty web: Emergent semantics through gossiping. In *Twelfth International World Wide Web Conference*, 2003.

[AK03] Rene J. Miller Anastasios Kementsietsidis, Marcelo Arenas. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *SIGMOD '03*, 2003.

[Ber03] Philip A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2003.

[BGK$^+$02] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proceedings of the WebDB Workshop*, 2002.

[DCvH$^+$02] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL web ontology language 1.0 reference, 2002. Manuscript available from `http://www.w3.org/2001/sw/WebOnt/`.

[DDH01] Anhai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: a machine learning approach. In *Proc. of SIGMOD*, 2001.

[DHW01] Denise Draper, Alon Y. Halevy, and Daniel S. Weld. The nimble integration system. In *Proc. of SIGMOD*, 2001.

[DMDH02] Anhai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proc. of the Int. WWW Conf.*, 2002.

[DR02] Hong-Hai Do and Erhard Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proc. of VLDB*, 2002.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[HC03] Bin He and Kevin Chen-Chuan Chang. Statistical schema integration across the deep web. In *Proc. of SIGMOD*, 2003.

[HED$^+$03] Alon Halevy, Oren Etzioni, Anhai Doan, Zachary Ives, Jayant Madhavan, Luke McDowell, and Igor Tatarinov. Crossing the structure chasm. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[HIM$^+$04] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The piazza peer-data management system. *Transactions on Knowledge and Data Engineering, Special issue on Peer-dta management, to appear*, 2004.

[HIST03] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *Proc. of ICDE*, 2003.

[HITM03] Alon Halevy, Zachary Ives, Igor Tatarinov, and Peter Mork. Piazza: Data management infrastructure for semantic web applications. In *Proc. of the Int. WWW Conf.*, 2003.

[HM03] Alon Y. Halevy and Jayant Madhavan. Corpus-based knowledge representation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1567–1572, 2003.

[KEW01] Cody Kwok, Oren Etzioni, and Dan Weld. Scaling question answering to the web. In *Proc. of the Int. WWW Conf.*, pages 150–161, 2001.

[KN03] Jaewoo Kang and Jeffrey Naughton. On schema matching with opaque column names and data values. In *Proc. of SIGMOD*, 2003.

[KNO$^+$02] P. Kalnis, W. Ng, B. Ooi, D. Papadias, and K. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proc. of SIGMOD*, 2002.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, 2002.

[MBC$^+$03] Jayant Madhavan, Philip Bernstein, Kuang Chen, Alon Halevy, and Pradeep Shenoy. Matching schemas by learning from others. In *Working notes of the IJCAI-03 workshop on Data Integration on the Web*, 2003.

[MBDH02] Jayant Madhavan, Philip Bernstein, Pedro Domingos, and Alon Halevy. Representing and reasoning about mappings between domain models. In *Proceedings of AAAI*, 2002.

[MBR01] Jayant Madhavan, Phil Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[MEH$^+$03] Luke McDowell, Oren Etzioni, Alon Halevy, Henry Levy, Steven Gribble, William Pentney, Deepak Verma, and Stani Vlasseva. Enticing ordinary people onto the semantic web via instant gratification. In *Proceedings of the Second International Conference on the Semantic Web*, October 2003.

[MH03] Jayant Madhavan and Alon Halevy. Composing mappings among data sources. In *Proc. of VLDB*, 2003.

[MQS03] http://www-3.ibm.com/software/ts/mqseries/, 2003.

[NM02] Natalya Noy and Mark A. Musen. PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. In *Proceedings of AAAI*, 2002.

[NOTZ03] Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. PeerDB: A P2P-based system for distributed data sharing. In *SIGMOD '03*, 2003.

[NWQ$^+$02] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjorn Naeve, Mikael Nilsson, Matthias Palmer, and Tore Risch. EDUTELLA: A P2P networking infrastructure based on RDF. In *Eleventh International World Wide Web Conference*, pages 604–615, 2002.

[PG02] Neoklis Polyzotis and Minos N. Garofalkis. Statistical synopses for graph-structured XML databases. In *SIGMOD '02*, 2002.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[RFQ$^+$02] Dragomir R. Radev, Weiguo Fan, Hong Qi, Harris Wu, and Amardeep Grewal. Probabilistic question answering on the web. In *Proc. of the Int. WWW Conf.*, pages 408–419, 2002.

[SBG$^+$03] S. Sizov, M. Biwer, J. Graupmann, S. Siersdorfer, M. Theobald, G. Weikum, and P. Zimmer. The BINGO! system for information portal generation and expert web search. In *Proceedings of the First Biannual Conference on Innovative Data Systems Research (CIDR), Asilomar(CA)*, 2003.

Database Exploration and Bellman

Theodore Johnson, Amit Marathe, Tamraparni Dasu
AT&T Labs – Research
{johnsont,marathe,tamr}@research.att.com

Abstract

Large industrial-scale databases tend to be poorly structured, dirty, and very confusing. There are many reasons for this disorder, not the least of which is that the application domains themselves are poorly structured, dirty and confusing. As data analysts, we are often called upon to mine, clean, or otherwise analyze these databases. In this article, we describe the types of problems we have encountered, tools and techniques we have developed to address these problems, and directions for future work.

1 Introduction

As analysts at a very large information-intensive corporation, we are often called upon to analyze business data sets. The analysis tasks range from data mining and analysis to assisting subject matter experts in data quality improvement projects. The source data might be a set of extracted views delivered in delimited ASCII format, or ODBC access to a dozen very large and complex databases which participate in a business process.

The most difficult aspect of these projects is to make sense of the available data. Because of the enormous scope and complexity of AT&T's services and offerings, these data sets would be challenging even if the data were clean, well-organized, and well-documented. In general, they are not, and we spend a significant portion of our time extracting metadata from the data set and from subject matter experts.

In this paper, we describe the kinds of problems we encounter in the data sets and the tools and techniques that we have developed to overcome these problems. We conclude with directions for future research.

2 Problems with Data Interpretation

Understanding the structure and relationships in databases with thousands of tables is a difficult task even in the best of conditions. We often encounter two problems which make the job even harder: poor documentation and data quality problems.

Metadata When we get access to a data set for an analysis, we almost always discover that the documentation and metadata is incomplete, inaccurate, or just plain missing. Sometimes the data set is provided as a collection of delimited ASCII files (or perhaps spreadsheet files). In this case the metadata is often just the file names and the field names. These data sets are either kept using an informal database (e.g. a spreadsheet), or are generated as a query against a database. In either case we need to depend on the data provider to write documentation for us — which is unlikely. A special problem occurs when the data is a query result: the query is a crucial piece of metadata. However the query(s) which generated the data are usually not provided (and often the provider is reluctant to reveal the query).

In other cases, we are provided with logins and programmatic access (e.g., ODBC) to the source databases (or we are given a copy of the database backup). A commercial DBMS provides many facilities for recording metadata. In addition to declaring keys and foreign key joins, we can examine which fields are indexed and what domains, views, and stored procedures are defined. There might also be explanatory comments associated with tables and fields. From the declared keys and foreign key joins, most DBMSs have tools which draw charts of the database structure.

Unfortunately, these metadata are usually incomplete. DBAs rarely label tables with comments, and often do not record keys, foreign key joins, and so on. There are several reasons for this lack of documentation. One is that creating documentation is tedious and unrewarded. The DBA knows what the table means (at least, she does when it is created), and why should she train her replacement? Enforcing key and foreign key join properties can create a high overhead, and can also be dangerous. It can be easier to turn off consistency constraints to make a data set load than it is to debug a data problem. Also, we have encountered serious data quality problems which were caused by the unintended consequences of enforcing foreign key join constraints. Another factor is that enterprise databases change frequently (i.e., because the modeled enterprise does), so the recorded metadata quickly becomes out-of-date. This problem is another disincentive to recording metadata — since its likely to be useless or misleading when the DBA needs it, why should she record it?

There is another reason why we usually find the metadata to be lacking. We are called upon to perform new data mining and data quality studies, and therefore are making new uses of the data. Common uses of the data are likely to be well documented, unusual uses aren't.

Even if the conventional metadata were complete, it usually does not provide enough information to interpret the data set. Some issues are:

- In textbooks, field names fully describe the field contents. The "natural join" makes a strong use of this property — since fields with the same name mean the same thing, the join condition between two tables is a test for equality between all fields with the same name. In practice, there are many names for fields with values which describe the same things. Sometimes the heterogeneity is the result of small misspellings, sometimes the field names are completely different. Often, fields with the same name have very different contents (e.g. *Address* in the IP_CIRCUIT table versus *Address* in the CUSTOMER table). Even if their values are drawn from the same domain, the values might refer to very different things (e.g., customer *Name* vs. salesperson *Name*). When dealing with multiple databases, these problems are even worse.

- Databases often show a surprising degree of heterogeneity. We expect this kind of problem when integrating multiple databases, but we encounter it even in a single database. A very large (1000 table or more) database is often run by multiple DBAs and might refer to multiple logical entities. For example, one group of tables might refer to customer accounts while another group of tables refers to invoices. Names, keys, formats, conventions, normalization, and so on might be significantly different between these different areas. Another cause of heterogeneity is the continual change in the enterprise. Although the DBAs might design a schema which perfectly fits the current state of the enterprise, within six months the database will be called upon to support entirely new types of customers and services. These are often shoehorned into existing tables to the greatest degree possible. As a result, a table will often have multiple disjoint sets of join paths, depending on the flavor of the tuple to be joined. We often find fields with very heterogeneous contents (e.g., describing a customer by name or by numeric key).

- The data in these databases is not static, rather it used (or is generated) by some enterprise processes. A proper interpretation of the data requires an understanding of these processes. However processes are often even less documented than the database.

Data Quality Problems Compounding the difficulties cause by missing, incomplete, or ambiguous metadata are data quality problems. We often find that a task which should be simple winds up being quite difficult because of poor data quality. We need to diagnose the extent of the problems, then explore the database to determine workaround procedures. A selection of the kinds of problems that we frequently encounter are:

Missing Data: We often find that important data does not exist in the database. Sometimes relevant fields are sparsely populated or completely missing. For example, we might want to find predictors of high-revenue customers. In this case we are certain to find that the *Revenue* field of the CUSTOMER table is almost completely NULL. If there is a CUSTOMER_REVENUE table, it is certain to have only a few dozen records. Perhaps the table is the result of a query and the DBA uses an inner join where an outer join would be more appropriate.

Incorrect Values: Even if we were to find a well-populated *Revenue* field, it is quite likely to contain incorrect data (or to be calculated in a way different that what we expect).

Poor Join Paths: Many times join paths are of poor quality – the foreign key join constraints are not maintained. So if we have well-populated CUSTOMER and REVENUE tables, its likely that we can join these tables for only a small fraction of the customers.

Duplicates in Key Fields: Finding a key for a table can be surprisingly difficult. Sometimes a key field (or fields) will contain duplicates. This problem occurs for many reasons. Sometimes unique keys get recycled (e.g., telephone numbers) and the table contains both old and current entities. The data might be the result of a query in which a join creates multiple entries for a few records. Perhaps we are trying to write an ad-hoc query and use a serendipitous join path.

In our experience, problems that we at first attribute to "poor quality data" often turn out to be problems in our interpretation. The CUSTOMER_REVENUE might be built only for customers of a particular class. The *Revenue* field might not include rebates or other pricing adjustments. The CUSTOMER table might refer to corporations as a while, while the REVENUE table refers to entities within corporations. To join these tables, we might need to use a table which associates entities to their parent corporations (we might have a few successful matches in the case of small businesses).

3 Tools and Techniques

Since we are naturally lazy, we have developed tools and techniques to make our jobs easier. The two that we discuss here are *database browsing* tools and *data quality metrics*. Although they are quite different, these tools have two things in common: they cut through the complexity of large systems, and they extract missing metadata.

3.1 Database Browsing

Finding relevant data is difficult in very large (1000+ tables) complex databases. This task is even more difficult when the data is dispersed among a dozen such databases. We found that we need advanced database browsing tools which could help us understand what the database contents and structure are. For example, we usually find "surprises" in the database, and we need to explore the database to determine if the surprise is a data quality problem or a problem in our understanding of the database structure.

The tool we developed, Bellman, *profiles* the database [5] to develop compact statistical summaries of the database contents. When a user browses a database, Bellman uses these summaries to provide informative displays and powerful exploration queries in an interactive fashion.

Many of the facilities that Bellman provides are fairly simple, but surprisingly useful. Bellman collects and interactively displays the number of rows in each table, and for each field the number of unique and null values. This simple function immediately tells about missing data. Bellman also collects information about multiple databases, and can manage multiple open connections to them. While Bellman does not yet support federated queries, the ability to browse and query multiple databases simultaneously is very useful. Other simple but useful tools are a collection of canned queries for sampling, summarizing, and visualizing a table's contents.

Some tools use more sophisticated algorithms, developed by others. Bellman will find all minimal approximate keys of up to four fields on all tables using an algorithm based in part on Tane [7]. We also collect the most frequent values of a field ("heavy hitters") using a sketch-based algorithm [1]. If a table has say 100 fields, we compute the most common field values using a one pass algorithm instead of 100 group-by queries. Most frequent value information is very useful because it gives a feel for the contents of a field, and also because default NULL values are usually the most frequent values (often there is more than one default NULL value). We are currently experimenting with using recently developed approximate string matching algorithms [6] to implement an approximate join engine.

We developed a crucial technology for Bellman, as is described in [3]. For every field, we can interactively compute the *resemblance* of the set of field values (resemblance is the size of the set intersection divided by the size of the set union). We use the resemblance find join paths, and to find data which is "similar" to a sample. With augmented information, we can obtain good estimates of join sizes and whether the join is one-to-one, one-to-many, or many-to-many. Many times two fields will match only after a small transformation. To help in the search for these pairs, we also provide tools for interactive substring similarity queries.

A typical use of Bellman would be as follows. We are asked to make a study of some particular business process. We obtain logins and ODBC access to the relevant databases, then use Bellman to collect database profiles. By talking to Subject Matter Experts (SMEs), we get an understanding of the business process and what our data targets are. Typically, we run into various problems — data is missing, join paths don't work, data mining analyses yield suspicious results, and so on. We begin a database exploration process to find solutions to these problems. We might want to find all data related to frame relay circuits, in which case we will search for frame relay circuit IDs by starting with a sample and using Bellman's resemblance and substring similarity tools. When we find these fields, we use Bellman to explore these tables to determine whether the information they contain is relevant to our needs. We might want to find alternative join paths, either because the documented join paths are broken or because we want a check on suspicious data (i.e., we want correlating information). We use the approximate keys and field resemblance facilities to find the alternative join paths, and the general browsing facilities to evaluate the results.

3.2 Data Quality Metrics

Data quality metrics are intended to assure the data user of the confidence he or she can place in the data. In general, data quality metrics measure the reliability and usability of the data. However, they can also serve to document critical metadata as well. The brief discussion here is taken from a recent book [2].

Conventionally, metrics have been designed for *static* views of the data, with time related aspects incorporated only tangentially, e.g., timeliness of the data. These metrics emphasize rigid constraints such as completeness, accuracy and consistency. In reality, such constraints are often not enforceable. Verifying accuracy would entail expensive inventory checking. Proxies such as sampling or tracking trouble tickets might be insufficient with a potential for built-in bias. Consistency and uniqueness might be impossible to ensure. For instance, a "customer" could mean a billing entity, a corporation, a contract of services , or a piece of equipment. Uniqueness is a stumbling block if different systems require representing an entity with slight variations e.g., "New

Vernon, NJ and “New Vernon NJ. This is especially true of complex legacy systems, which have all kinds of quirks hard coded into the system.

Modern uses and data applications require a much more *dynamic* definition of metrics to reflect accurate conformance to process flows. For instance, there has been a recent focus on auditing databases that support business operation for data quality issues. Such an audit often reveals flaws in the process, in addition to any glitches the data. Based on this, we can define two categories of data quality metrics, — *operational* and *diagnostic*. Operational metrics identify unexpected divergence of process from specification, providing guidance for fixing flaws that manifest as data quality issues. Diagnostic metrics on the other hand, merely identify the flaws in the data and do not provide insight into how to fix them. Completeness is a good example. “The data set has ten percent missing data” is informative but not useful for finding the problem that is causing the data quality issue.

As mentioned earlier, the lack of metadata is a major issue in a data analysis project. Here, we make a distinction between access and interpretation. *Accessibility* can be measured by seemingly bureaucratic criteria such as number of calls, number of people contacted, level of escalation and total time taken from request for access to data to actual physical access to data. *Interpretability* is more nuanced and harder to measure. The schema might be accurate, but there might be hidden rules that are not documented. For example, the people who created the data might have informally compressed the data with rules such as “if the resources allocated to a customer have consecutive serial numbers, record the first and last serial number and set attribute Flag to Y”. We cannot interpret the data if we do not know the rule. Another example in this context is that of censoring in duration data. A call beyond 100 minutes might be censored, so that all we know is that it lasted beyond 100, we cannot distinguish between calls of 101 minutes and 10001 minutes. Such rules are seldom documented and are revealed only by browsing the data and validating the findings with SMEs. Hidden rules show up as glitches in analysis (unexpected spikes in histograms) and data integration (only 10% of the records matched) . The number of analytical passes needed to get reasonable results is a good metric to measure interpretability. Similarly in the operations setting, the number of manual interventions needed (measures automation) and the proportion of data that flows through the process successfully (end-to-end completion) are good metrics. Automation is particularly important since many data quality glitches arise from manual intervention and workaround patches to modify a process designed to automatically complete a multi-step task or transaction. A longer discussion of data quality metrics can be found in [2].

Implementing data quality metrics in a data quality program involves three major stages: (a) data preparation (gathering, storing, integration, manipulation etc.), (b) gathering domain knowledge through interaction with SMEs, e.g., build business rules, devise appropriate metrics and (c) validating the data against the rules and quantifying data quality using the metrics. The three stages are iterated until the experts and the users sign off on the usability of the data. However, the second stage is surprisingly challenging. Subject matter experts are hard to identify (scattered across the company, personnel and project transitions) and often have no incentive to share their time or knowledge. Even if they cooperate, different experts frequently disagree on definitions and details. Finally, there is often a disconnect between the articulation of the rules by the experts and the implementation by the technical staff. A potential solution is offered in [4], where rule based programs are used to encode the expert’s knowledge in a form that can be easily understood and verified by the expert. Such an approach also allows for representing knowledge that is acquired piecemeal, out of sequence and that needs to be updated frequently.

Despite the challenges of implementing data quality metrics as a part of a data quality or data auditing program, there are significant benefits. First, the process of defining data quality metrics results in the creation of metadata and documentation about the data and the process. Second, it puts in place a mechanism for conducting frequent end-to-end audits to catch data quality issues as soon as they arise before they get entrenched and corrupt large sections of the data. Finally, it increases the confidence in the data by putting in place objective, clearly defined metrics that are transparent and cannot be manipulated easily for political ends.

4 Conclusions

The tools that we have developed are very useful in our analysis tasks, letting us quickly resolve seemingly difficult problems. We have found that some of the simplest tools are often the most effective. However, we feel that additional, more sophisticated tools are needed:

- Database structure mining, for example to find clusters of inter-related tables.
- Incorporating dynamic information, such as that extracted from query logs. However, we have found that DBAs are (often justifiably) reluctant to share these logs.
- Integrating SME metadata with extracted metadata.

The digital revolution has only started, the volume and complexity of enterprise information will continue to increase. Software engineers have faced the problem of developing very large scale interoperating code, and in response have developed many design, analysis, and reverse engineering tools and techniques. Analogously, database engineers need similar tool suites.

References

[1] G. Carmode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proc. Principles of Database Systems*, pages 296–306, 2003.

[2] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley, 2003.

[3] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. ACM SIGMOD Conf.*, pages 240–251, 2002.

[4] T. Dasu, G. Vesonder, and J. Wright. Data quality through knowledge engineering. In *Prof. Conf. Knowledge Discovery and Data Mining*, pages 705–710, 2003.

[5] Evoke Software. Data profiling and mapping, the essential first step in data migration and integration projects. http://www.evokesoftware.com/pdf/wtpprDPM.pdf, 2000.

[6] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *Proc. Intl. world Wide Web Conf.*, pages 90–101, 2003.

[7] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional dependencies and approximate dependencies using partitions. In *Proc. IEEE Intl. Conf. on Data Engineering*, pages 392–401, 1998.

On Schema Discovery[*]

Renée J. Miller Periklis Andritsos
Department of Computer Science
University of Toronto
{miller,periklis}@cs.toronto.edu

Abstract

Structured data is distinguished from unstructured data by the presence of a schema describing the logical structure and semantics of the data. The schema is the means through which we understand and query the underlying data. The schema permits the more sophisticated structured queries that are not possible over schema-less data. Most systems assume that the schema is predefined and is an accurate reflection of the data. This assumption is often not valid in networked databases that may contain data originating from many sources and may not be valid within legacy databases where the semantics of data have evolved over time. As a result, querying and tasks that depend on structured queries (including data integration and schema mapping) may not be effective. In this paper, we consider the problem of discovering schemas from data. We focus on discovering properties of data that can be exploited in querying and transforming data. Finally, we very briefly consider the suitability of mining approaches to the task of schema discovery.

1 Introduction

As the size, number, and complexity of databases continues to grow, understanding their structure and semantics gets more difficult. This, together with other associated problems such as the lack of documentation or the unavailability of the original database designers can make the task of understanding the structure and semantics of databases a very difficult one. At the same time, data sources may contain errors and may contain data that was integrated from many sources. This integration may introduce additional anomalies or duplicate data values and records. Fewer and fewer databases contain only raw data that has been put in electronic form for the first time and that has been carefully designed and structured for a well-defined application task. Rather, modern databases contain data that has been integrated and transformed, often many times, from other source(s).

No matter how principled or sophisticated the integration approach, we simply cannot always guarantee a perfect integration. Similarly, no matter how carefully a database was designed in the past, the data semantics may still evolve or errors may be introduced into the data. Hence, the data may be inconsistent or incomplete with respect to its schema (where the schema may include structuring primitives such as table declarations along with constraints). There are many forms of data inconsistency including incorrect or erroneous data values. A

[*]Funding provided by NSERC, MITACS, a Premier's Research Excellence Award, and a gift from Microsoft.

data instance is typically said to be inconsistent if it does not conform to the constraints of the schema [ABC99]. Similarly, there are many forms of incompleteness including missing or unknown values or missing records [Gra02]. As with inconsistency, incompleteness is usually studied with respect to a schema. Notice that the assumption is that we trust the schema, that is, the schema is an accurate model of the time-invariant properties of the data. However, in both legacy databases (where the semantics of the data may have evolved since the schema was designed) and in integrated data (where data with different semantics may be added) this may not be a valid assumption.

In this paper, we briefly consider a different approach. Rather than viewing the data as being inconsistent or incomplete with respect to a given schema, we consider the schema to be potentially inconsistent or incomplete with respect to a given data instance. With this view, we consider whether we can propose techniques for discovering a better schema for the data. Of course, given a single data instance, we cannot be sure that the properties we find will hold for all possible instances (that is, we cannot be sure that these are time invariant properties). But we believe that automated techniques can be used to suggest potential schemas and that such suggestions can be incorporated into physical and logical data design and integration tools.

Below, we consider how having a schema that is inaccurate with respect to a given data instance can hamper both querying and the integration process itself. We then consider, in closer detail, the properties of schemas with an eye toward developing algorithms for schema discovery. We present one application of these ideas involving the clustering of tuples and values based on their structure.

2 Schema Properties

We use the term schema to refer to a set of structuring primitives (predicates), for example, relations or nested-relations, and a set of constraints (logical statements) over these primitives. Examples of constraints include key constraints and referential constraints. Constraints play an important role in data design and, as we will argue below, they are integral in understanding what forms of learning approaches are most useful for schema discovery. Before discussing the discovery problem, we examine in closer detail the role of schemas in providing structural and semantic information about data. We also consider the role schemas have played in the integration process.

Structured queries are by definition typed, that is, they take instances of a fixed schema (or type) and create an instance of a fixed schema [AHV95].[1] When schemas are inaccurate, queries can produce unexpected results. Consider the following schema and instance.

title	director	actor	genre	release date
Godfather II	M. Scorsese	R. De Niro	Crime	1974
Good Fellas	F. F. Coppola	R. De Niro	Crime	1998
Vertigo	A. Hitchcock	J. Stewart	Thriller	1958
N by NW	A. Hitchcock	C. Grant	-	1959
Alexander the Great	Baz Luhrmann	-	-	-
Water	Deepa Mehta	-	-	-
My Life Without Me	Isabel Coixet	-	-	-

Table 1: An instance of the movie relation

Intuitively, the first four tuples of this relation correspond to movies that have already been released and the last three tuples to movies that are currently being shot. We could imagine a situation where Table 1 has

[1]Of course, this analysis excludes work on higher-order or polymorphic queries [LSS01] which produces instances conforming to a family of schemas.

been produced after integrating two separate relations, one for released movies and another for pre-release films (which may include movies in production or movies that may never be shot or released) or a situation where the original data requirements included only released movies, but later applications required information about pre-release movies. In either case, the schema asserts that movies may have release dates and genres. A query analyzing trends in the genre of movies may give very misleading results as this information is systematically missing for pre-released films. Similarly, schemas in which constraints hold but are not expressed can lead to query results that are much different than anticipated. In our movie database, films may be associated with many companies which are responsible for various aspects of their production, financing, and release. However, films produced under the old studio system may all be associated with exactly one company. We can view such schemas as being incomplete (or more generally inaccurate) in that they fail to model important logical distinctions.

In addition to the effect on queries, inaccurate schemas can influence the integration process. Traditional integration approaches were very schema-centric [BLN86]. Given a set of schemas, these approaches would create an integrated schema or view. In the above example, it is not apparent from the schema that the movie relation contains two different types of movies. Hence, the integration of this database with another in which movies are separated by their production stage may result in all movies from our database being incorrectly assumed to belong to a single production stage (for example, they may all be assumed to be released movies). In more recent work, the focus of integration methods has shifted away from the problem of creating integrated schemas, to the problem of creating queries between independently created schemas [MHH00, HMN$^+$99]. Such approaches have been called query-centric [Ull03], to distinguish them from view-centric approaches which use views as a means of integrating data and modeling the query capabilities of different sources [LRO96, Hal01]. Query-centric and view-centric approaches have been combined in approaches that create more general mappings (sometimes called GLAV, global-and-local-as-view, or BAV, both-as-view, mappings) [Len02, PVM$^+$02, MP02]. However, because these structured queries and views are inherently typed, the effectiveness of all of these approaches depends to a large extent on the quality of the schemas being used. Logical semantics that are not modeled in a schema cannot be exploited. Mappings created over inaccurate schemas, such as the schema in Table 1, may incorrectly integrate or transform data (for example, by treating release and pre-released movies as a homogeneous collection). These mappings may produce unexpected or undesired results including (possibly) instances that do not conform to the target schema of the mappings.

In schema discovery, we will seek to discover logical structure that may be useful for integration or querying. Before presenting examples of discovery techniques, we consider one additional property of schemas that will be useful in designing and understanding schema discovery methods.

Schemas, like structured query languages that use them, treat data values largely as uninterpreted objects. This property has been called *genericity* [AHV95] and is closely tied to *data independence*, the concept that schemas should provide an abstraction of a data set that is independent of the internal representation of the data. That is, the choice of a specific data value (perhaps “Coppola” or “F.F.C.” or “francis ford coppola” in our example) has no inherent semantics and no influence on the schema used to structure director values. The semantics captured by a schema is independent of such choices.[2]

For query languages, genericity is usually formalized by saying that a query must commute with all possible permutations of data values (where the permutations may be restricted to preserve a distinguished set of constants) [AHV95]. Similarly, we can formalize the genericity of schemas in the following way.

Definition 1: A schema S is said to be *generic* if for all instances I of S and for all permutations π of values in I, $\pi(I)$ is also an instance of S.

It is possible in SQL and other languages used in practice, to write non-generic queries (for example, using

[2]The term *generic* is sometimes used to mean *data model independent*. In this paper, we will use the adjective *generic* to refer to schemas that exhibit the genericity property.

user-defined functions) and non-generic constraints (for example, `check release-date < 2004`). However, such constraints are not foundational to structuring data and when used, are typically restricted to domains with a known, application-specific semantics (such as the ordering of years in this example). A similar constraint on movie titles (`check title < 2001`) is likely to be meaningless with respect to the semantics of movie data. Our thesis is not that such application-specific information is unimportant, but rather that it is often unavailable or inconsistent across different data sources which may use different conventions for encoding data values. Our techniques will therefore focus on the discovery of generic structure and constraints.

3 Schema Discovery Techniques

Schema discovery is not a new problem. Early approaches were motivated by the observation that an important source of inaccuracy in relational schemas is incompleteness, particularly in the constraints. Techniques for enumerating constraints that hold on a relational database include algorithms for finding dependencies (such as functional and multivalued dependencies) [BB95, Bel95, Bel97, KMRS92, WBX98]. Given an instance of a schema, the idea is to find all constraints within a specified class that describe the data. Of course, from a single database instance, we cannot determine whether the constraint is a dependency, that is, whether it will continue to hold as the data changes. However, these techniques can find a set of candidate dependendencies. Furthermore, several approaches consider the constraint mining problem over dirty data and propose techniques to find *approximate* dependencies that hold for most records [HKPT98]. The candidate dependencies found by such algorithms can be examined by a user or provided as input to a data design tool.

Constraint mining has applications beyond schema discovery (for example, to semantic query optimization). But certainly discovered constraints can be used to find "better" schemas. If additional constraints are found, then we can apply normalization methods to produce a (vertically) decomposed schema. Such methods are well-known to produce better schemas by reducing redundancy in the data and removing (or lessening) such evils as update anomalies. Notice that such a discovery approach operates by finding redundancy (specifically the redundancy that can be characterized by traditional relational dependencies) and removing this redundancy. Also, all of the constraint mining techniques we have mentioned are generic in that they do not rely on any application-specific semantics for the representation of data values.

Constraint mining looks to find a set of constraints (logical statements) that hold on a given data set and structure. The set of predicates used in these constraints is fixed – where the set of predicates corresponds to the set of tables defined in the schema. In our example, this means that the algorithms will search for constraints that hold over all movies and will not consider separating movie tuples into different tables and finding constraints that hold within each table. The re-structuring that is done is with respect to attributes. Attributes are regrouped into (potentially overlapping) sets, each representing a new table.

Now consider the problem of producing better schemas even when the set of predicates is not fixed. Conceptually, we would like to find a clustering of tuples such that each cluster can be described by a good schema. So we wish to find redundancy in the data, but redundancy that can be removed by partitioning tuples horizontally rather than vertically. Our approach is based on a clustering method called LIMBO [ATMS03]. Clustering produces a partitioning of objects where objects within the same cluster are similar and objects in different clusters are dissimilar. To apply clustering to this problem, we have to define a distance metric for relational tuples that is generic. In our movie example, it is not obvious how to measure the similarity (or dissimilarity) of the values "Coppola", and "Scorsese". Nor is it obvious how to define the distance between tuples "Vertigo" and "Good Fellas". If we are seeking to remove redundancy, intuitively we need a measure of similarity that reflects the redundancy in these tuples. However, even with such a measure, it is not obvious how to define a quality measure for the clustering. Yet, for humans there is an intuitive notion of quality for clustering. A good clustering is one where the clusters are *informative* about the tuples they contain. Since tuples are expressed over attribute values, we require that the clusters be informative about the attribute values of the tuples they hold. That is, given a cluster, we should be able to predict the attribute values of the tuples in the cluster to the greatest degree possible. Hence, highly redundant tuples will tend to be grouped together. The quality measure of the clustering is the information that the clusters hold about the attributes. Since a clustering is a summary of the data, some information may be lost. Our objective is to minimize this loss, or equivalently to minimize the increase in uncertainty.

Consider again the data in Table 1 and the partitioning of tuples into two clusters. Clustering C groups the first four movies together into one cluster, c_1, and the remaining three into another, c_2. Note that cluster c_2 preserves all information about the actor, genre and release date of the movies it holds. For cluster c_2, we know with certainty that these values are missing (or null). For cluster c_1, we have lost some information about these attributes so we have less certainty. But we do know, for example, that there is a 50% chance that the genre is "Crime". In this example, any other clustering will result in greater information loss. To see this, consider a clustering D again with two clusters. The first cluster d_1 contains the first three tuples and the second cluster d_2 contains the last four (so unlike in C, the fourth tuple has be assigned to the second cluster d_2). In d_2, we maintain certainty about the value of genre, but we lose information about actor and release date.

We have formalized this intuition and developed a scalable algorithm for clustering large categorical data sets [ATMS03]. We have applied our techniques to clustering both relational data and web data. Furthermore, LIMBO is a hierarchical algorithm that produces clusterings for a large range of k values (where k is the number of clusters). This property is very important for schema discovery since we can evaluate different k values (among the many clusterings produced in a single application of the algorithm) to determine one that achieves the best schema design.

We have argued that schema discovery involves characterizing (and eliminating) redundancy in the data. Our approach to characterizing redundancy is similar in spirit to recent work on using information theory to both characterize good schema designs and to compare the quality of different designs [AL03] (and to work characterizing the information theoretic properties of dependencies [DR00]). However, this work defines the information content of database elements with respect to a fixed set of constraints. Furthermore, the measure is restricted to valid database instances that satisfy these constraints. In our work, we have characterized the information content independent of constraints since such constraints are often unknown (or only partially known) and the data may contain errors. Nonetheless, there are important synergies between these results.

4 Conclusions

Schemas are typically the result of a data design process (performed by a human designer or by an automated tool). The choices made in data design are known to be highly subjective. A schema is inherently one out of many possible choices for modeling a data set. To understand and reconcile heterogeneous data, we may need to understand (and explicitly represent) some of the alternative design choices. Our current research focuses on the development of schema discovery techniques for finding such alternative designs. Our long term research goal is to find generic methods for discovering schemas that fit the data. Our work on LIMBO is a first step toward this goal. This work is part of a broader research agenda designed to develop solutions for creating, managing and transforming structure and meta-data, including schemas and mappings between schemas [AFF+02].

References

[ABC99] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[AFF+02] P. Andritsos, R. Fagin, A. Fuxman, L. M. Haas, M. Hernandez, C.-T. Ho, A. Kementsietsidis, R. J. Miller, F. Naumann, L. Popa, Y. Velegrakis, C. Vilarem, and L.-L. Yan. Schema Management. *Data Engineering Bulletin*, 25(3), September 2002.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[AL03] M. Arenas and L. Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. In *PODS*, pp. 15-26, 2003.

[ATMS03] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik. Limbo: Scalable Clustering of Categorical Data. Hellenic Database Symposium, 2003.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[BB95] Siegfried Bell and Peter Brockhausen. Discovery of constraints and data dependencies in relational databases. In Nada Lavrač and Stefan Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *LNAI*, pages 267–270, Berlin, April 1995. Springer.

[Bel95] S. Bell. Discovery and maintenance of functional dependencies by independencies. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 27–32. AAAI Press, 1995.

[Bel97] Siegfried Bell. Dependency mining in relational databases. In Dov M. Gabbay, Rudolf Kruse, Andreas Nonnengart, and Hans Jürgen Ohlbach, editors, *Proceedings of the First International Joint Conference on Qualitative and Quantitative Practical Reasoning*, volume 1244 of *LNAI*, pages 16–29, Berlin, June9–12 1997. Springer.

[DR00] M. M. Dalkilic and E. L. Robertson. Information Dependencies. In *PODS*, Dallas, TX, USA, 2000.

[Gra02] G. Grahne. Information Integration and Incomplete Information. *IEEE Data Engineering Bulletin*, 25(3):46–52, 2002.

[HMN$^+$99] L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz, and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering*, 22(1):31–36, 1999.

[Hal01] A. Y. Halevy. Answering Queries Using Views: A Survey. *The Int'l Journal on Very Large Data Bases*, 10(4):270–294, 2001.

[HKPT98] Y. Huhtala, J. Kärkkäinen, P. Porkka and H. Toivonen. Efficient Discovery of Functional and Approximate Dependencies Using Partitions. *Int'l Conf. on Data Engineering*, 392–401, 1998.

[KMRS92] M. Kantola, H. Mannila, K.-J. Rih, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *International Journal of Intelligent Systems*, 7(7):591–607, September 1992.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.

[LSS01] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Trans. on Database Sys. (TODS)*, 26(4):476–519, 2001.

[MHH00] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 77–88, Cairo, Egypt, September 2000.

[MP02] Peter McBrien and Alexandra Poulovassilis. Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. In *Conference on Advanced Information Systems Engineering (CAISE)*, pages 484–499, 2002.

[PVM$^+$02] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 598–609, Hong Kong SAR, China, August 2002.

[Ull03] J. D. Ullman. Where Is Database Research Headed? Keynote. In *IEEE Conference on Database Systems for Advanced Applications (DASFAA)*, Kyoto, Japan, 2003.

[WBX98] S. K. M. Wong, C. J. Butz and Y. Xiang. Automated Database Schema Design Using Mined Data Dependencies. *Journal of the American Society for Information Science*, 49(5), pages 455–470, April 1998.

Sponsored by the IEEE Computer Society

CALL FOR PARTICIPATION

20th International Conference on Data Engineering

March 30 - April 2, 2004
Omni Parker House Hotel, Boston, USA

http://www.cse.uconn.edu/icde04

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 20th International Conference on Data Engineering will be held in Boston, Massachusetts, USA -- an academic and technological center with a variety of historical and cultural attractions of international prominence within walking distance.

The ICDE 2004 International Conference on Data Engineering provides a premier forum for:

- sharing research solutions to problems of today's information society;
- exposing practicing engineers to evolving research, tools, and practices and providing them with an early opportunity to evaluate these;
- raising awareness in the research community of the problems of practical applications of data engineering;
- promoting the exchange of data engineering technologies and experience among researchers and practicing engineers;
- identifying new issues and directions for future research and development work.

ICDE 2004 invites research submissions on all topics related to data engineering, including but not limited to those listed below:

1. Indexing, access methods, data structures
2. Query processing (standard and adaptive) and query optimization
3. Data Warehouse, OLAP, and Statistical DBs
4. Mining Data, Text, and the Web
5. Semi-structured data, metadata, and XML
6. Web Data Management
7. Middleware, workflow, and security
8. Stream processing, continuous queries, and sensor DB's
9. Database applications and experiences
10. Distributed, parallel and mobile DB's
11. Temporal, Spatial and Multimedia databases
12. Scientific and Biological DBs; Bioinformatics

PROGRAM

- 63 research papers from 441 full submissions
- Three plenary speakers
- Four panels
- Three 3-hour seminars, four 1.5-hour seminars
- 21 poster papers
- 12 industrial papers
- 16 demos

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398