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

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*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 $\mathcal{S}_{\mathcal{G}}$ and $\mathcal{S}_{\mathcal{F}}$, respectively. Observe that both $\mathcal{S}_{\mathcal{G}}$ and $\mathcal{S}_{\mathcal{F}}$ contain the initial input sequences in I. This is to ensure that the MDL module has a wide range of DTDs

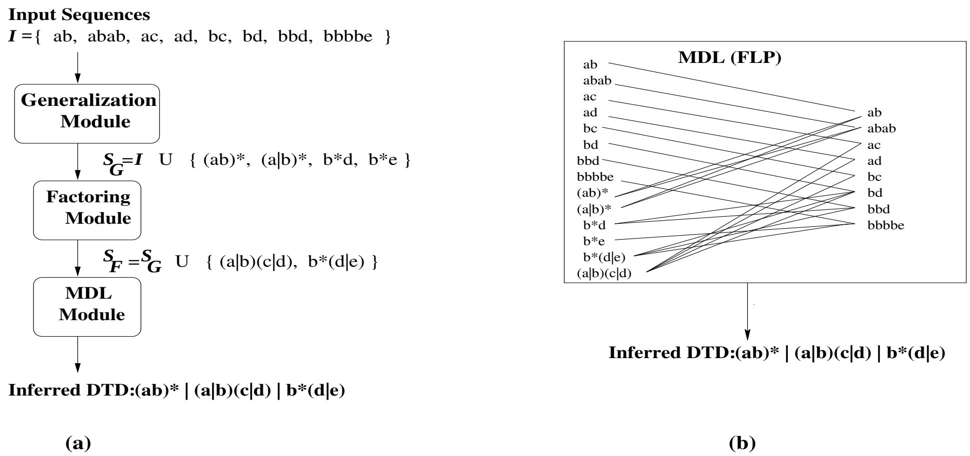

Figure 2: (a) XTRACT System Architecture. (b) XTRACT's MDL Subsystem.

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.