Finding ID Attributes in XML Documents

Denilson Barbosa and Alberto O. Mendelzon

University of Toronto
Toronto, Ontario, Canada
{dmb,mendel}@cs.toronto.edu

Abstract. We consider the problem of discovering candidate ID and IDREF attributes in a schemaless XML document. We characterize the complexity of the problem, propose a heuristic algorithm for the problem, and discuss experimental results.

1 Introduction

XML documents may be accompanied by schemas that specify their structure, and impose restrictions on the *values* of some elements or attributes. The WWW Consortium has defined two schema formalisms for XML: Document Type Definitions (DTDs) [13] and XML Schema [14]. Although these languages differ in several aspects, both allow the specification of *ID*, *IDREF* and *IDREFS* attributes. ID attributes are unique identifiers for the elements that bear them; IDREF attributes are logical pointers to ID attributes; IDREFS attributes are pointers to *sets* of ID attributes. IDREF(S) attributes establish references among elements in the document, turning the XML trees into graphs. XML query languages have ways of "navigating" these graphs seamlessly via tree edges and reference edges (see [1]). In a sense, ID attributes serve as keys for the elements, while IDREF(S) attributes serve as foreign keys. However, as we shall see, there are some significant differences between ID/IDREF(S) attributes and keys.

Schemas are important tools that enable data exchange, efficient storage, and query formulation. Because many XML databases are likely to be missing their schemas [8], there has been interest in the literature in the problem of *inferring* a schema for a document. Current approaches, however, focus on extracting the portions of the schema that define the tree structure of the document. In this paper, we consider the problem of discovering the portions of the schema that constrain the non-tree edges in the graph, that is, discovering candidate ID and IDREF attributes in a schemaless document.

1.1 Related Work

To the best of our knowledge, there is no previous work on the problem of discovering ID and IDREF attributes for XML documents.

Grahne and Zhu [6] consider the problem of finding approximate keys in XML documents. That work differs from ours in the following ways. First, a key is only required to be unique over the set of elements of the same type, while ID attributes are required by the XML specification to be unique over the entire document. Second, the keys considered in [6] are not necessarily unary, as are ID attributes. Finally, that work does not consider finding foreign keys, the equivalent to IDREF attributes.

Garofalakis *et al.* [5] consider the problem of extracting DTDs from XML documents. However, the authors consider only structural constraints, and do not deal with attribute values.

Buneman *et al.* [3] define keys for XML. That work lays the foundations and also the basic notation used here and in [6]. Arenas *et al.* [2] study the interaction of DTDs with keys and foreign-keys for XML.

A similar problem in the relational setting is finding keys and foreign keys from a set of relation instances (see, e.g., Kantola *et al.* [7]). That problem differs from ours in two ways: foreign keys are typed, while IDREF(S) attributes are not; and the scope of a key is a relation, while, as we mentioned above, the scope of ID attributes is the entire document.

Organization of the Paper. Section 2 introduces the machinery we will use throughout the paper. The semantics of ID and IDREF attributes in XML, and how they are represented in our notation are covered in Section 3. The complexity of finding a good set of ID attributes is given in Section 4; a heuristic algorithm for this problem is given in Section 5. We consider the easier problem of finding IDREF(S) attributes Section 6. An experimental validation of our algorithms is presented in Section 7. We conclude in Section 8.

2 Preliminaries

2.1 Data Model

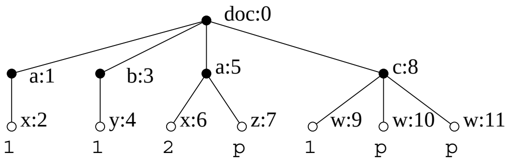
We represent XML documents as labeled trees with three kinds of nodes for representing elements, attributes and textual content. Each document D has a distinguished node called $root_D$. For clarity, we deal with a single document and drop the subscripts for identifying documents. Each node v in the tree has a label, denoted $label(v)$; a unique identifier (*id* for short), denoted $id(v)$; and a value, denoted $value(v)$. Without loss of generality, we ignore the ordering of nodes in the tree. Let $\mathcal{I}$ be set of node *id*s and let $\mathcal{V}$ be the set of all values in the document.

We say two nodes v_1 and v_2 are *node equal*, written $v_1 =_n v_2$, if $id(v_1) = id(v_2)$, and *value equal*, written $v_1 =_v v_2$, if $value(v_1) = value(v_2)$.

Running Example. We will use the simple XML document shown in Figure 1(a) to illustrate the various concepts throughout the paper. Figure 1(b) shows the tree representation of our example XML document. Each node v is annotated with a pair $label(v) : id(v)$. Multivalued attributes are represented as multiple attribute nodes, all with the same label. We also show the values of the attribute nodes in the document.

```
<doc>
    <a x='1'/>
    <b y='1'/>
    <a x='2' z='p'/>
    <c w='1 p p'/>
</doc>
```

(a) XML document.



(b) Tree representation.

Fig. 1. Example XML document and its corresponding tree representation.

2.2 Path Expressions

We use a subset of XPath, defined as follows. Let Σ^E and Σ^A be the sets of all element and, respectively, attribute labels that occur in the document; let $\Sigma = \Sigma^E \cup \Sigma^A$. A path expression is a string in the grammar $E ::= \varepsilon \mid a \mid \_ \mid E* \mid E.E$, where ε is a path expression that matches the empty path; $a \in \Sigma$; "_" is a path expression that matches any symbol in Σ; "*" is the usual Kleene star operator; and "." is the usual concatenation operation.

Let x be a node in the tree. $x[\![P]\!]$ denotes the set of nodes (i.e., node *id*s) reached by starting at node x and following paths that conform to P. We shall use $[\![P]\!]$ as an abbreviation for $root[\![P]\!]$.

2.3 Keys

Keys for XML documents were studied in [3]. A key specification consists of: a set of elements (called the *target set*) for which the key holds; and a set of *key paths* that specify the values on which the key is defined. In the following, Q defines a target set and $P_1, \ldots, P_k$ are key paths:

Definition 1 ([3]) *A node x satisfies a key specification $(Q, \{P_1, \ldots, P_k\})$ iff for any two nodes y_1, y_2 in $x[\![Q]\!]$, if for all $i, 1 \le i \le k$ there exist $z_1 \in y_1[\![P_i]\!]$ and $z_2 \in y_2[\![P_i]\!]$, such that $z_1 =_v z_2$, then $y_1 =_n y_2$.*

2.4 Attribute Mappings

Attributes in XML documents are not independent entities; rather, they are part of the elements where they are declared. The following definition ties together elements and attributes, and is central to the paper.

Definition 2 *For $x \in \Sigma^E$ and $y \in \Sigma^A$, we call the* attribute mapping *of y over x, denoted by M_x^y, the $\mathcal{I} \times \mathcal{I}$ relation defined by*

$$M_x^y = \{(z, w) : z \in [\![\_*.x]\!], w \in z[\![y]\!]\}$$

That is, the relation M_x^y contains edges in the tree that connect element nodes labeled x and attribute nodes labeled y.

We apply the standard relational projection operator to attribute mappings. For instance, $\pi_E(M_x^y)$ is the set of all unique *id*s for all elements in $[\![\_*.x]\!]$. Similarly for $\pi_A(M_x^y)$.

It is worth mentioning that, in [13], elements of the same label have the same *type*. Thus, we will use the term element type and element label interchangeably. We define the *type* of an attribute mapping, denoted by τ, as $\tau(M_x^y) = x$.

Example 1 The document in Figure 1(a) has the following non-empty attribute mappings: $M_a^x = \{(1,2),(5,6)\}$, $M_a^z = \{(5,7)\}$, $M_b^y = \{(3,4)\}$, and $M_c^w = \{(8,9),(8,10),(8,11)\}$.

As we discuss in the next section, the semantics of ID attributes relates element *types* to attribute *values*. Thus, we define the following.

Definition 3 *The* image *of attribute mapping m, denoted by $\iota(m)$ is the $\mathcal{I} \times \mathcal{V}$ relation defined as*

$$\iota(m) = \{z : z = value(w), w \in \pi_A(m)\}$$

Example 2 The images of the attribute mappings in Example 1 are: $\iota(M_a^x) = \{\texttt{1}, \texttt{2}\}$, $\iota(M_b^y) = \{\texttt{1}\}$, $\iota(M_a^z) = \{\texttt{p}\}$, and $\iota(M_c^w) = \{\texttt{1}, \texttt{p}\}$.

3 ID Attributes

The semantics of ID attributes in XML are defined as follows [13, Section 3.3.1][14, Section 3.3.8]:

– (ID1) Validity constraint: ID
Values of type ID must be single-valued. A name must not appear more than once in an XML document as a value of this type; *i.e.*, ID values must uniquely identify the elements which bear them.
– (ID2) Validity constraint: One ID per Element Type
No element type may have more than one ID attribute specified.

Rule (ID1) above can be split into two conditions: (ID1.a) the values of ID attributes must be singletons; (ID1.b) no two elements in the document may have the same value for their ID attribute. Note that an element may occur without its ID attribute, if the attribute is declared as *IMPLIED* instead of *REQUIRED*. [13, Section 3.3.2]

Definition 4 *An attribute mapping m is a* candidate ID mapping *if it is an injective function, that is,*

$$|m| = |\pi_E(m)| = |\pi_A(m)| = |\iota(m)|$$

Example 3 Among the attribute mappings in Example 1, M_a^x, M_b^y and M_a^z are candidate ID mappings.

Because of rule (ID1.b), we cannot say when a single attribute can be used as an ID attribute; we need to do it for *sets* of attributes, as follows.

Definition 5 (ID Set) *A set of attribute mappings $I = \{m_1, \ldots, m_n\}$ is an* ID set *for an XML document iff: each $m_i \in C$ is a candidate ID mapping,*

$$\bigcap_{m_i \in I} \tau(m_i) = \varnothing, \textit{ and } \bigcap_{m_i \in I} \iota(m_i) = \varnothing$$

Example 4 Consider the attribute mappings in Example 1. The following are all ID sets: $I_1 = \{M_a^x\}$, $I_2 = \{M_b^y\}$, $I_3 = \{M_a^z\}$, and $I_4 = \{M_a^z, M_b^y\}$. Obviously, no ID set can contain both M_a^x and M_a^z. Also, no ID set can contain both M_a^x and M_b^y because the images of these mappings share the symbol `1`.

ID Attributes are not Unary Keys. Note that ID attributes are different from unary keys. Defining one unary key for each ID attribute in the DTD is not enough. To illustrate this, note that the document in Figure 1(a) satisfies keys $(a, \{x\})$ and $(b, \{y\})$. However, that document does not satisfy a DTD that defines x as ID attribute of a and y as ID attribute of b, because the value 1 appears in both attributes.

We could define a single unary key of the form $(\_*, \{x\})$, where x is some attribute label, and the target set is all elements in the document. This is suggested in [3]. However, doing so requires all ID attributes to have the same label (namely, x). Note that the document in Figure 1(a) satisfies a DTD that defines z as ID attribute for a and x as ID attribute for b; however, we cannot represent both ID attributes as a single unary key, unless we rename either x or z. Furthermore, a document satisfies a unary key of the form $(\_*, \{x\})$ only if all attributes labeled x are in fact ID attributes (e.g., one could not have an IDREF(S) attribute labeled x). Neither restriction is consistent with [13].

Finally, note that, unlike primary keys in the relational setting, that are restricted to be non-null, an ID attribute can be missing; i.e., if M_x^y is an ID attribute mapping, there can be elements of type x in the document that do not define y attributes.

3.1 IDREF and IDREFS Attributes

The semantics of IDREF and IDREFS attributes in XML are much simpler than the semantics of ID attributes [13, Section 3.3.1]:

– Validity constraint: IDREF
  Values of type IDREF *must* be single-valued, and values of type IDREFS *may* be multi-valued; each value must match the value of an ID attribute on some element in the XML document.

Given an ID set I, a mapping m that represents an IDREF attribute is such that $\iota(m) \subseteq \bigcup_{p \in I} \iota(p)$. If m contains multivalued attributes, then m represents an IDREFS attribute.

Evidently, the set of mappings that define IDREF(S) attributes depends on the ID set for the document:

Example 5 Consider the document in Figure 1(a) and the ID sets shown in Example 4. The only ID sets for which there are IDREF(S) attributes are $I_1 = \{M_a^x\}$ (M_b^y represents an IDREF attribute), and $I_4 = \{M_a^z, M_b^y\}$ (M_c^w represents an IDREFS attribute).

3.2 Finding "Good" ID Attributes

The example above shows that in general there will be multiple ID sets for a given document. How do we choose the "best" set? A simple approach is to select a set that contains as many ID attributes as possible; that is, an ID set of maximum cardinality. More generally, we allow attribute mappings to have weights, and choose ID sets of maximum total weight. The weight of a mapping m may depend both on the number of elements that are assigned assigns unique identifiers by m and on the number of reference edges that m induces in the XML graph.

4 The Complexity of Computing ID Sets

In this section we give the complexity of finding an ID set of maximum weight (for maximum cardinality ID sets, it suffices to fix the weights of all mappings to be unitary). For simplicity, and without loss of generality, we take as input a set of candidate ID mappings instead of a document; note that the mappings can be easily computed from the document. We consider initially the decision version of the problem, which we will call K-IDSET:

K-IDSET:
Instance: a set of candidate ID mappings $C = \{m_1, \ldots, m_n\}$ and an integer $K \leq n$.
Question: is there $I \subseteq C$ such that I is an ID set and $|I| \geq K$?

Theorem 1 *K-IDSET is NP-Complete.*

Proof. The problem is trivially in NP. We show completeness by a reduction from INDEPENDENT SET (IS). Let $G = (V, E)$ be a simple connected graph with vertex set $V = \{v_1, \ldots, v_n\}$, and edge set $E = \{e_1, \ldots, e_m\}$. We define the attribute mappings as follows. Let $\mathcal{I} = V \cup E$, and define $value(x) = x,\ x \in \mathcal{I}$. For each vertex $v_i \in V$, we create a mapping $m_i = \{(v_i, e_j) : e_j \in E \text{ is incident on } v_i\}$, and define $\tau(m_i) = v_i$; let $C = \{m_1, \ldots, m_n\}$ be the set of all such mappings. It is clear that G has an independent set of size K iff C has an ID set of size K. Also, C can be constructed in time polynomial on $n + m$.

The optimization version of the problem is a maximization problem defined as follows:

MAX-IDSET:
Instance: a set of candidate ID mappings $C = \{m_1, \ldots, m_n\}$ and a weight function $w : 2^{\mathcal{I} \times \mathcal{I}} \rightarrow \mathbb{Q}^+$. **Goal**: find $I \subseteq C$ such that I is an ID set and there is no $I' \subseteq C$ such that I' is an ID set and $\sum_{m_i \in I'} w(m_i) > \sum_{m_j \in I} w(m_j)$.

As Theorem 1 shows, K-IDSET contains IS as a subproblem. This is in fact bad news for the design of an algorithm for MAX-IDSET; the problem turns out to be complete for the class of NP-hard optimization problems, a notion called $\mathbf{FP}^{\mathbf{NP}}$-completeness [9].

Theorem 2 MAX-IDSET *is* $\mathbf{FP}^{\mathbf{NP}}$*-Complete. Furthermore, unless* $\mathbf{P} = \mathbf{NP}$, MAX-IDSET *has no constant factor approximation algorithm.*

Proof. The MAX INDEPENDENT SET (MIS) is the optimization version of IS; we prove completeness by reduction from MIS. The reduction is essentially the same given for the proof of Theorem 1, with the added restriction that $w(m_i) = w(v_i)$, $v_i \in V$. It is known that a constant factor approximation algorithm for MIS would yield a polynomial time algorithm for IS [4]. Since the reduction given in the proof above is in fact an L-reduction (see [9]), any constant factor approximation algorithm for MAX-IDSET is also a constant factor approximation algorithm for MIS.

The weight function in the maximization problem captures the notion of "goodness" of a candidate ID set and can be any function. As long as w can be computed efficiently (say, in polynomial time), the complexity of the problem does not change.

Theorem 2 essentially tells us that traditional techniques for obtaining approximability bounds (e.g., [12]) for this problem are of no use here. Instead, a heuristic that produces feasible solutions is, in a sense, as good as any other. For more details about MIS and related problems, we refer the reader to [4, 9], and also to a recent survey [10].

5 A Heuristic Algorithm for Computing ID sets

In this section we give a greedy heuristic for the MAX-IDSET problem. Let $\mathtt{M} = \{m_1, \ldots, m_n\}$ be the set of all non-empty attribute mappings in the document. The "goodness" of a mapping will be captured by the following two definitions.

Definition 6 *The* support *of attribute mapping* m *is defined as*

$$\phi(m) = |m| / \sum_{p \in \mathtt{M}} |p|$$

Definition 7 *The* coverage *of attribute mapping* m *is defined as*

$$\chi(m) = \left(\sum_{p \in \mathtt{M}, p \neq m} |\iota(m) \cap \iota(p)| \right) / \sum_{p \in \mathtt{M}} |\iota(p)|$$

```
Input: XML document $X$
1. $M \leftarrow$ all attribute mappings in $X$; $C \leftarrow$ all candidate ID mappings in $M$
     sorted by decreasing size;
2. Compute the weight $w(m)$ of each $m$ in $C$
3. for $t$ in $\Sigma^E$ do :
4.     Let $m$ be a highest-weight mapping of type $t$ in $C$
5.     Remove from $C$ all mappings of type $t$ except $m$
6. end for
7. for $m$ in $C$ do :
8.     $S \leftarrow$ all mappings in $C$ whose images intersect $\iota(m)$
9.     if $w(m) > \sum_{p \in S} w(p)$ then remove all $p \in S$ from $C$; otherwise remove $m$ from $C$
10. end for
11. ID set $\leftarrow C$
```

Fig. 2. Algorithm for computing ID sets.

Note that $\phi(m) \leq 1$, and $\chi(m) \leq 1$.

The support of attribute mapping M_x^y is the fraction of edges in the XML tree that connect x elements to y attributes. The coverage of an attribute mapping measures how much of the image of that mapping occurs elsewhere, as a fraction of all mappings images in the document. Intuitively, a candidate ID mapping with high support represents an attribute that identifies a high proportion of the occurrences of some element type, while a mapping with high coverage represents an attribute that many other attributes refer to; these properties capture different aspects of being a good candidate for an ID attribute.

Figure 2 shows our greedy algorithm for computing ID sets. Initially, we find all attribute mappings, and sort those that are candidate ID mappings by decreasing size. Next, we assign weights to all mappings. Recall that C is an ID set only if for any two mappings $m_1, m_2 \in C$, we have no *type conflict* (i.e., $\tau(m_1) = \tau(m_2)$) and no *image conflict* (i.e., $\iota(m_1) = \iota(m_2)$). Mappings pruned (i.e. removed from C) in an iteration of the algorithm are never considered in later iterations. We deal with type conflicts first (lines 3-6). From all mappings of type t, we choose one of highest weight (breaking ties arbitrarily) and prune all others; we repeat this for all element types. Image conflicts are dealt with next (lines 7-10). For each mapping m, we find the set of mappings that conflict with its image (denoted by S), and prune either m or all mappings in S depending on the the weight of m and the aggregated weight of S. We use the weight function $w(m) = \alpha \cdot \chi(m) + \beta \cdot \phi(m)$, where α and β determine the relative priority of choosing mappings with high support (i.e., mappings that assign identifiers to more elements) over mappings with high coverage (i.e., mappings that yield more refences between elements).

Correctness. We now show that the output of the algorithm in Figure 2 is always an ID set. First, it is easy to see that lines 3-6 remove all type conflicts that exist in the document. And since the mappings pruned in that phase are not considered further, we need only to show that lines 7-10 remove all image conflicts left in C. Each iteration of this loop removes at least one mapping that causes conflicts in C. We can also show that in each iteration, no mapping in S has been visited in a previous iteration; it follows that all image conflicts are eventually eliminated.

Complexity. We consider the complexity of building the data structures first. Recall that Definition 4 is based on the *sizes* of the relations involved. Also, only the *images* of the mappings are required for finding conflicts, and for computing coverages. Thus, for each $m \in \mathtt{M}$ we only materialize $\iota(m)$, which is kept sorted for faster processing; we also maintain the *values* of $|m|$, $|\pi_E(m)|$, $|\pi_A(m)|$, $|\iota(m)|$, and $w(m)$. Since $|\mathtt{M}| = O(n)$, where n is the size of the document, the space cost of the algorithm is $O(n)$. Note that all the data structures can be built in a single scan of the document.

Now we consider computing the weights of the mappings. Note that, for mapping m, $|m| = O(n)$. Computing $\chi(m)$ requires finding the intersection of $\iota(m)$ with all other mappings in $\mathtt{M}$ (note that mappings in $\mathtt{M} - C$ could represent IDREF attributes). The intersection of two mapping images can be found in $O(n \log n)$ time. Hence, the time to compute $\chi(m)$ is $O(n^2 \log n)$. Computing $\phi(m)$ can be done in $O(\log n)$; thus, line 2 of the algorithm requires $O(n^3 \log n)$ time.

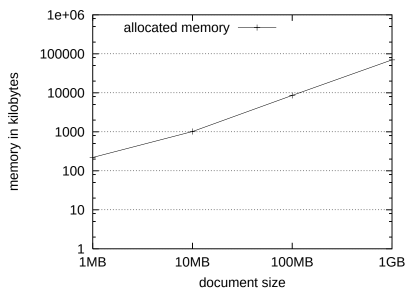
Eliminating type conflicts (lines 3-6) requires finding all conflicting mappings for type t, which can be done in $O(n \log n)$ time; and finding one mapping of highest weight. Thus, lines 3-6 require $O(n^2 \log n)$ time. The complexity of eliminating image conflicts (lines 7-10) is as follows. Computing S is done by searching in C, and requires $O(n^2 \log n)$ time. We can compute the aggregate weight of S as we perform the search. Therefore, the total complexity of our algorithm is $O(n)$ space and $O(n^3 \log n)$ time.

5.1 Approximation Bound

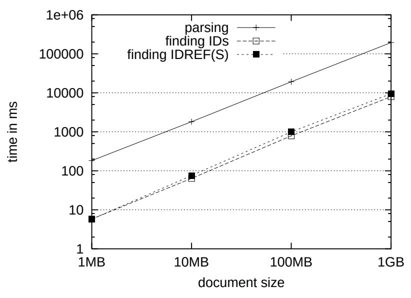
Recall that a constant factor approximation algorithm for this problem is unlikely to exist. The algorithm in Figure 2 is adapted from a well-known heuristic for the MIS problem [9, 10]. The following result can be shown. Let δ_m be the number of conflicts that exist for mapping m; let Δ be the largest δ_m in the document (note that Δ has a standard meaning in the graph-theoretic formulation of IS). The approximation factor of the algorithm in Figure 2 is $1/\Delta$. Furthermore, all known better algorithms for MIS fail to improve substantially on this bound [10].

6 Finding IDREF(S) Attributes

Once we have an ID set, finding IDREF attributes is easy. Let I be an ID set and m be an attribute mapping not in I. If m is the mapping of an IDREF attribute, then (1) $\iota(m) \subseteq \cup_{m_i \in I} \iota(m_i)$, and (2) $|\pi_A(m)| = |\pi_E(m)|$. If condition (1) holds and $|\pi_A(m)| > |\pi_E(m)|$, then m is a mapping of an IDREFS attribute.



(a) Memory requirements.



(b) Running times.

Fig. 3. Log-log plots of memory requirements and running times for XMark datasets of varying sizes. For the running times, each plot represents the average of of 30 trials.

Testing condition (1) above requires computing the intersection of m and all the mappings in I, and, thus, requires time $O(n^2 \log n)$ in the worst case. One way of doing so would be to actually materialize $\cup_{m_i \in I} \iota(m_i)$ and compute all intersections using this new relation. However, although IDREF(S) attributes are untyped, computing the pairwise intersection of m with each individual mapping in I can give extra knowledge about the structure of the document. Finally, testing condition (2) can be done in $O(\log n)$ time using our data structures.

7 Experimental Validation

In this section we present some results that show how our algorithm behaves in practice. Our implementation was done in C++ and used the Xerces SAX parser[1] (in non-validating mode) for scanning the documents and building our data structures. All experiments were run on a 2.4 GHz Pentium 4 machine with Linux and 1GB of memory. Our implementation accepts three parameters: α and β in the weight function; and μ which is a lower bound on the cardinality of the mappings considered. For these experiments, we fix $\alpha = \beta = 1$.

7.1 Scalability

Our first experiment shows how the algorithm scales with document size. We used four documents for the XMark benchmark [11] of varying sizes. Figure 3(a) shows the amount of memory used for representing the data structures for each of the documents. As for running times, Figure 3(b) shows separately the times spent on parsing, computing the ID set, and finding the IDREF(S) attributes based on the ID set chosen, for each of the documents.

[1] Available at `http://xml.apache.org`.

DTD	Element Rules	ID		IDREF		IDREFS	
		REQ.	IMPL.	REQ.	IMPL.	REQ.	IMPL.
XMark (4.2KB)	77	4	0	10	0	0	0
Mondial (4.3KB)	41	11	0	6	4	1	11
W3C DTD (47KB)	141	6	113	13	2	0	2
XDF (30KB)	142	0	11	0	19	1	0

Table 1. Characteristics of the DTDs used in our experiments. The table shows the number of element declaration rules, ID,IDREF and IDREFS attributes declared in each DTD.

Several observations can be made from Figure 3. First, as expected, both the memory requirements and the time for parsing grow linearly with the document size. Second, as far as resources are concerned, the algorithm seems viable: it can process a 10MB document in less than 2 seconds using little memory, on a standard PC. Finally, Figure 3(b) shows that the running time of the algorithm is clearly dominated by the parsing: as one can see, the parsing time is always one order of magnitude higher than any other operation. Of course this is due to the I/O operations performed during the parsing.

7.2 Quality of the Recommendations

The previous experiment showed that the algorithm has reasonable performance. We now discuss its effectiveness. We considered 11 DTDs for real documents found on a crawl of the XML Web (see [8]), for which we were able to find several relatively large documents, and compared the recommendations of our algorithm to the specifications in those DTDs. Due to space limitations, we discuss here the results with 3 real DTDs that illustrate well the behavior of our algorithm with real data: Mondial[2], a geographic database; a DTD for the XML versions of W3C recommendations[3]; and NASA's eXtensible Data Format (XDF)[4], which defines a format for astronomical data sets. We also report the results on the synthetic DTD used in the XMark benchmark.

Recall attributes are specified in DTDs by rules `<!ATTLIST` $e\ a\ t\ p$`>`, where e is an element type, a is an attribute label, t is the type of the attribute, and p is a participation constraint. Of course, we are interested in ID, IDREF and IDREFS types only; the participation constraint is either REQUIRED or IMPLIED. Table 1 shows the number of attributes of each type and participation constraint in the DTDs used in our experiments.

The DTDs for XDF and XML specifications are *generic*, in the sense that they are meant to capture a large class of widely varying documents. We were not able to find a single document that used all the rules in either DTD. The XMark and Mondial DTDs, on the other hand, describe specific documents.

[2] `http://www.informatik.uni-freiburg.de/~may/Mondial/florid-mondial.html`

[3] `http://www.w3.org/XML/1998/06/xmlspec-19990205.dtd`

[4] `http://xml.gsfc.nasa.gov/XDF/XDF_home.html`

Document	μ	$\lvert M \rvert$	Correct		Misclass.		Accuracy		Artifacts	
			ID	IDREF	ID	IDREF	ID	IDREF	ID	IDREF
XMark (10MB)	1	16	3	9	1	1	0.75	0.90	0	0
Mondial (1.2MB)	1	48	11	23	0	0	1.00	1.00	0	0
XML Schema Part 2 (479KB)	1	91	13	11	0	0	1.00	1.00	9	16
	2	69	12	10	1	1	0.92	0.91	5	10
	3	62	11	10	2	1	0.85	0.91	2	7
	4	61	11	10	2	1	0.85	0.91	2	7
	5	57	11	10	2	1	0.85	0.91	1	7
XDF document (38KB)	1	50	4	2	0	0	1.00	1.00	9	3
	2	40	3	1	1	0	0.75	0.50	6	0
	3	31	3	1	1	0	0.75	0.50	4	0

Table 2. Analysis of our algorithm on different documents. The table shows, for each document and value for μ: the number of mappings considered; the number of ID/IDREF attributes that were correctly classified; the number of ID/IDREF attributes that were misclassified; and the number of artifacts that were recommended as ID/IDREF attributes. The accuracy is defined as (Correct)/(Correct+Misclassifications).

Metrics. This section describes the metrics we use to compare the recommendations of our algorithm to the corresponding attribute declarations in the DTD. For participation constraints, if $\frac{|\pi_E(M_x^y)|}{|[\![\_*.x]\!]|} = 1$ we will say y is REQUIRED for x; otherwise, we say y is IMPLIED.

We consider two kinds of discrepancies between the recommendations made by the algorithm and the specifications in the DTDs: *misclassifications* and *artifacts*. A misclassification is a recommendation that does not agree with the DTD, and can occur for two reasons. First, there may be attributes described as ID or IDREF in the DTD but ignored by the algorithm. Second, there may be attributes specified as ID in the DTD but recommended as IDREF by the algorithm, or vice-versa.

A rule `<!ATTLIST` $e\ a\ t\ p$`>` is misclassified if the algorithm either does not recommend it or recommends it incorrectly, except if:

– e is declared optional and $[\![\_*.e]\!] = \varnothing$;
– a is declared IMPLIED and $\pi_A(M_e^a) = \varnothing$;
– a is an IDREFS attribute, $|\pi_E(M_e^a)| = |M_e^a|$, and our algorithm recommends it as IDREF.

Artifacts occur when the algorithm recommends attributes that do not appear in any rule in the DTD as either ID or IDREF. For example, an attribute that occurs only once in the document (e.g., at the root) might be recommended as an ID attribute. Artifacts may occur for a variety of reasons; it may be that an attribute serves as an ID for a particular document, but not for all, or that it was not included in the DTD because the user is not aware of or does not care about this attribute's properties.

Results. Table 2 compares the number of correct classifications to the number of misclassifications and artifacts produced for our test documents, all of which were valid according to the respective DTDs. For the W3C and XDF DTDs we ran the algorithm on several documents, and we report here representative results. For clarity, we report the measures and the accuracy scores for IDREF and IDREFS attributes together.

As one can see, our algorithm finds all ID and IDREF attributes for the Mondial DTD; also, no artifacts are produced for that document. The algorithm also performs very well for the XMark document. The misclassifications reported occur because there are two mappings with identical images in the document, and our algorithm picks the “wrong” one to be the ID attribute. We were not able to find all ID and IDREF attributes for the other two DTDs. However, this was expected, given that these DTDs are too broad and the instances we examined exercise only a fraction of the DTD rules. Note that the XDF DTD is roughly as large as the test document we used; in fact, most XDF documents we found were smaller than the XDF DTD.

Table 2 also shows a relatively high number of artifacts that are found by our algorithm, especially for the XDF DTD. Varying the minimum cardinality allowed for the mappings reduces the number of artifacts considerably; however, as expected, doing so also prunes some valid ID and IDREF mappings. It is interesting that some of the artifacts found appear to be natural candidates for being ID attributes. For example, one ID artifact for the XML Schema document contained email addresses of the authors of that document. Also, most of the IDREF artifacts refer to ID attributes that are correctly classified by our algorithm. For example, in the XML Schema document with $\mu = 2$, only 1 IDREF artifact refers to an ID artifact.

8 Conclusion

We discussed the problem of finding candidate ID and IDREF(S) attributes for schemaless XML documents. We showed this problem is complete for the class of NP-hard optimization problems, and that a constant rate approximation algorithm is unlikely to exist. We presented a greedy heuristic, and showed experimental results that indicate this heuristic performs well in practice.

We note that the algorithm presented here works in main memory, on a single document. This algorithm can be extended to deal with collections of documents by prefixing document identifiers to both element identifiers and attribute values. This would increase its resilience to artifacts and the confidence in the recommendations. Also, extending the algorithm to secondary memory should be straightforward.

As our experimental results show, our simple implementation can handle relatively large documents easily. Since the parsing of the documents dominates the running times, we believe that the algorithm could be used in an interactive tool which would perform the parsing once, and allow the user to try different ID sets (e.g., by requiring that certain attribute mappings be present/absent). This would allow the user to better understand the relationships in the document at hand.

Acknowledgments. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and Bell University Laboratories. D. Barbosa was supported in part by an IBM PhD. Fellowship. This work was partly done while D. Barbosa was visiting the OGI School of Science and Engineering.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kauffman, 1999.
2. M. Arenas, W. Fan, and L. Libkin. On verifying consistency of XML specifications. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 259–270, 2002.
3. P. Buneman, S. Davidson, W. Fan, C. Hara, and W.-C. Tan. Keys for XML . In *Proceedings of the Tenth International Conference on the World Wide Web*, pages 201–210. ACM Press, 2001.
4. M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, 1979.
5. M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A system for extracting document type descriptors from XML documents. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 165–176, Dallas, Texas, USA, May 16-18 2000.
6. G. Grahne and J. Zhu. Discovering approximate keys in XML data. In A. Press, editor, *Proceedings of the eleventh international conference on Information and knowledge management*, pages 453–460, McLean, Virginia, USA, November 4-9 2002.
7. M. Kantola, H. M. andK. J. Räihä, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7(7):591–607, September 1992.
8. L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a first study. In *Proceedings of The Twelfth International World Wide Web Conference*, 2003.
9. C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1995.
10. V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, June 1997.
11. A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, Hong Kong, China, August 2002.
12. V. Vazirani. *Approximation Algorithms*. Springer Verlag, 2003.
13. Extensible markup language (XML) 1.0 - second edition. W3C Recommendation, October 6 2000. Available at: `http://www.w3.org/TR/2000/REC-xml-20001006`.
14. XML Schema part 1: Structures. W3C Recommendation, May 2 2001. Available at: `http://www.w3.org/TR/xmlschema-1/`.