Validity-Sensitive Querying of XML Databases [⋆]

Slawomir Staworko and Jan Chomicki

University at Buffalo,
{staworko,chomicki}@cse.buffalo.edu

Abstract. We consider the problem of querying XML documents which are not valid with respect to given DTDs. We propose a framework for measuring the invalidity of XML documents and compactly representing minimal repairing scenarios. Furthermore, we present a validity-sensitive method of querying XML documents, which extracts more information from invalid XML documents than does the standard query evaluation. Finally, we provide experimental results which validate our approach.

1 Introduction

XML is rapidly becoming the standard format for the representation and exchange of semi-structured data (documents) over the Internet. In most contexts, documents are processed in the presence of a schema, typically a Document Type Definition (DTD) or an XML Schema. Although currently there exist various methods for maintaining the validity of semi-structured databases, many XML-based applications operate on data which is invalid with respect to a given schema. A document may be the result of integrating several documents of which some are not valid. Parts of an XML document could be imported from a document that is valid with respect to a schema slightly different than the given one. For example, the schemas may differ with respect to the constraints on the cardinalities of elements. The presence of legacy data sources may even result in situations where schema constraints cannot be enforced at all. Also, temporary violations of the schema may arise during the updating of an XML database in an incremental fashion or during data entry.

At the same time, DTDs and XML Schemas capture important information about the expected structure of an XML document. The way a user formulates queries in an XML query language is directly influenced by her knowledge of the schema. However, if the document is not valid, then the result of query evaluation may be insufficiently informative or may fail to conform to the expectations of the user.

Example 1. Consider the DTD D_0 in Figure 1 specifying a collection of project descriptions: Each project description consists of a name, a manager, a collection of subprojects, and a collection of employees involved in the project. The following query Q_0 computes the salaries of all employees that are not managers:

$$/projs//proj/name/emp/following\_sibling::emp/salary$$

[⋆] Research supported by NSF Grants IIS-0119186 and IIS-0307434.

```
<!ELEMENT projs  (proj*)>
<!ELEMENT proj   (name,emp,proj*,emp*)>
<!ELEMENT emp    (name,salary)>
<!ELEMENT name   (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
```

Fig. 1. DTD D_0

```
<projs><proj>
  <name> Cooking Pierogies </name>
  <proj>
    <name> Preparing Stuffing </name>
    <emp><name> John </name>
         <salary> 80K </salary></emp>
    <emp><name> Mary </name>
         <salary> 40K </salary></emp>
  </proj>
  <emp><name> Peter </name>
       <salary> 30K </salary></emp>
  <emp><name> Steve </name>
       <salary> 50K </salary></emp>
</proj></projs>
```

Fig. 2. An invalid document T_0.

Now consider the document T_0 in Figure 2 which lacks the information about the manager of the main project. Such a document can be the result of the main project not having the manager assigned yet or the manager being changed.

The standard evaluation of the query Q_0 will yield the salaries of *Mary* and *Steve*. However, knowing the DTD D_0, we can determine that an `emp` element following the `name` element `''Cooking Pierogies''` is likely to be missing, and conclude that the salary of *Peter* should also be returned.

Our research addresses the impact of invalidity of XML documents on the result of query evaluation. The problem of querying invalid XML documents has been addressed in the literature in two different ways: through *query modification* or through *document repair*. Query modification involves various techniques of distorting, relaxing, or approximating queries [14, 21, 3]. Document repair involves techniques of cleaning and correcting databases [9, 17]. Our approach follows the second direction, document repair, by adapting the framework of *repairs* and *consistent query answers* developed in the context of inconsistent relational databases [4]. A *repair* is a consistent database instance which is *minimally* different from the original database. Various different notions of minimality have been studied, e.g., set- or cardinality-based minimality. A *consistent query answer* is an answer obtained in *every* repair. The framework of [4] is used as a foundation for most of the work in the area of querying inconsistent databases (for recent developments see [8, 12]).

In our approach, differences between XML documents are captured using sequences of atomic operations on documents: inserting/deleting a leaf. Such operations are used in the context of incremental integrity maintenance of XML documents [1, 5, 6] (modification of a node's label is also handled but we omit it because of space limitations). We define *repairs* to be valid documents obtained from a given invalid document using sequences of atomic operations of *minimum cost*, where the cost is measured simply as the number of operations. Valid answers are defined analogously to consistent answers. We consider schemas of XML documents defined using DTDs.

Example 2. The validity of the document T_1 from Example 1 can be restored in two alternative ways:

1. by inserting in the main project a missing emp element (together with its subelements name and salary, and two text elements). The cost is 5.
2. by deleting the main project node and all its subelements. The cost is 19.

Because of the minimum-cost requirement, only the first way leads to a repair. Therefore, the valid answers to Q_0 consist of the salaries of *Mary*, *Steve*, and *Peter*.

In our opinion, the set of atomic document operations proposed here is sufficient for the correction of *local* violations of validity created by missing or superfluous nodes. The notion of valid query answer provides a way to query possibly invalid XML documents in a *validity-sensitive* way. It is an open question if other sets of operations can be used to effectively query XML documents in a similar fashion.

The contributions of this paper include:

– A framework for validity-sensitive querying of such documents based on measuring the invalidity of XML documents;
– The notion of a *trace graph* which is a compact representation of all repairs of a possibly invalid XML document;
– Efficient algorithms, based on the trace graph, for the computation of valid query answers to a broad class of queries;
– Experimental evaluation of the proposed algorithms.

Because of space limitations we omitted the proofs of most of the theorems. These will be included in a forthcoming technical report.

2 Basic definitions

In our paper we use a model of XML documents and DTDs similar to those commonly used in the literature [5, 6, 16, 19].

Ordered labeled trees We view XML documents as labeled ordered trees with text values. For simplicity we ignore attributes: they can be easily simulated using text values. By Σ we denote a fixed (and finite) set of tree node labels and we distinguish a label $\mathtt{PCDATA} \in \Sigma$ to identify *text nodes*. A text node has no children and is additionally labeled with an element from an infinite domain Γ of text constants. For clarity of presentation, we use capital letters $\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \ldots$ for elements from Σ and capital letters $X, Y, Z \ldots$ for variables ranging over Σ.

We assume that the data structure used to store a document allows for any given node to get its label, its parent, its first child, and its following sibling in time $O(1)$. For the purpose of presentation, we represent trees as terms over the signature $\Sigma \setminus \{\mathtt{PCDATA}\}$ with constants from Γ.

Example 3. The tree T_1 from Figure 3 can be represented with the term $\mathtt{C}(\mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{B}())$.

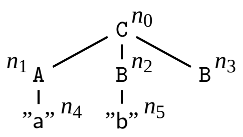


Fig. 3. A running example.

DTDs For simplicity our view of DTDs omits the specification of the root label. A *DTD* is a function D that maps labels from $\Sigma \setminus \{PCDATA\}$ to regular expressions over Σ. The size of D, denoted $|D|$, is the sum of the lengths of the regular expressions occurring in D.

A tree $T = X(T_1,\ldots,T_n)$ is *valid* w.r.t. a DTD D if: (1) T_i is valid w.r.t. D for every i and, (2) if $X_1,\ldots,X_n$ are labels of root nodes of $T_1,\ldots,T_n$ respectively and $E = D(X)$, then $X_1 \cdots X_n \in L(E)$.

Example 4. Consider the DTD $D_1(\mathtt{A}) = \mathtt{PCDATA} + \varepsilon$, $D_1(\mathtt{B}) = \varepsilon$, $D_1(\mathtt{C}) = (\mathtt{A}\cdot\mathtt{B})^*$. The tree $\mathtt{C}(\mathtt{A}(\mathtt{a}),\mathtt{B}(\mathtt{b}),\mathtt{B}())$ is not valid w.r.t. D_1 but the tree $\mathtt{C}(\mathtt{A}(\mathtt{a}),\mathtt{B}())$ is.

To recognize strings satisfying regular expressions we use the standard notion of *non-deterministic finite automaton* (NDFA) [15] $M = \langle \Sigma, S, q_0, \Delta, F\rangle$, where S is a finite set of *states*, $q_0 \in S$ is a distinguished *starting state*, $F \subseteq S$ is the set of *final states*, and $\Delta \subseteq S \times \Sigma \times S$ is the *transition relation*.

2.1 Tree edit distance and repairs

Tree operations A *location* is a sequence of natural numbers defined as follows: ε is the location of the root node, and $v \cdot i$ is the location of i-th child of the node at location v. This notion allows us to identify nodes without fixing a tree.

We consider two *atomic tree operations* (or *operations* for short) commonly used in the context of managing XML document integrity [1, 5, 6]:

1. *Deleting* a leaf at a specified location.
2. *Inserting* a leaf at a specified location. If the tree has already a node at this location, we shift the existing node to the right together with any following siblings.

We note that our approach can be easily extended to handle the operation of *Modifying* the label of a node (omitted here because of space limitations). We use sequences of editing operations to transform the documents. The *cost* of a sequence of operations is defined to be its length, i.e., the number of operations performed when applying the sequence. Two sequences of operations are *equivalent* on a tree T if their application to T yields the same tree. We observe that some sequences may perform redundant operations, for instance inserting a leaf and then removing it. Because we focus on finding cheapest sequences of operations, we restrict our considerations to redundancy-free sequences (those for which there is no equivalent but cheaper sequence).

Note that a deletion (an insertion) of a whole subtree can be performed with a sequence of deletions (resp. insertions) of length equal to the size of the tree.

Definition 1 (Edit distance). *Given two trees T and S, the* edit distance $dist(T,S)$ *between T and S is the minimum cost of transforming T into S.*

Note that the distance between two documents is a metric, i.e. it is positively defined, symmetric, and satisfies the triangle inequality.

For a DTD D and a (possibly invalid) tree T, a sequence of operations is a sequence *repairing* T w.r.t. D if the document resulting from applying the sequence to T is valid w.r.t. D. We are interested in the cheapest repairing sequences of T.

Definition 2 (Distance to a DTD). *Given a document T and a DTD D, the* distance $dist(T,D)$ *of T to D is the minimum cost of repairing T, i.e.*

$$dist(T,D) = \min\{dist(T,S) | S \text{ is valid w.r.t } D\}.$$

Repairs The notions of distance introduced above allow us to capture the minimality of change required to repair a document.

Definition 3 (Repair). *Given a document T and a DTD D, a document T' is a* repair *of T w.r.t. D if T' is valid w.r.t. D and $dist(T,T') = dist(T,D)$.*

Note that, if repairing a document involves inserting a text node, the corresponding text label can have infinitely many values, and thus in general there can be infinitely many repairs. However, as shown in the following example, even if the operations are restricted to deletions there can be an exponential number of non-isomorphic repairs of a given document.

Example 5. Suppose we work with documents labeled only with $\Sigma = \{\mathtt{A},\mathtt{B},\mathtt{T},\mathtt{F}\}$ and consider the following DTD: $D(\mathtt{A}) = \mathtt{T}\cdot\mathtt{A} + \mathtt{A}\cdot\mathtt{F} + \mathtt{B}\cdot\mathtt{B}$, $(\mathtt{B}) = \varepsilon$, $D(\mathtt{T}) = \varepsilon$, $D(\mathtt{F}) = \varepsilon$. The tree $\mathtt{A}(\mathtt{T}(),\mathtt{A}(\ldots\mathtt{A}(\mathtt{T}(),\mathtt{A}(\mathtt{B}(),\mathtt{B}()),\mathtt{F}())\ldots),\mathtt{F}())$ consisting of $3n+2$ elements has 2^{n-1} repairs w.r.t. D.

3 Computing the edit distance

In this section we present an efficient algorithm for computing the distance $dist(T,D)$ between a document T and a DTD D. The algorithm works in a bottom-up fashion: we compute the distance between a node and the DTD after finding the distance between the DTD and every child of the node.

3.1 Macro operations

Now, we fix a DTD D and a tree $T = X(T_1,\ldots,T_n)$. The base case, when T is a leaf, is handled by taking $n = 0$. We assume that the values $dist(T_i,D)$ have been computed earlier. We recall that the value $dist(T_i,D)$ is the minimum cost of a sequence of atomic tree operations that transforms T_i into a valid tree. Similarly, the value $dist(T,D)$ corresponds to the cheapest sequence *repairing* T. We model the process of repairing T with 3 *macro operations* applied to the root of T:

1. *Deleting* a subtree rooted at a child.
2. *Repairing* recursively the subtree rooted at a child.
3. *Inserting* as a child a minimum-size valid tree whose root's label is Y for some $Y \in \Sigma$.

Each of these macro operations can be translated to a sequence of atomic tree operations. In the case of a repairing operation there can be an exponential number of possible translations (see Example 5), however, for the purpose of computing $dist(T,D)$ we only need to know their cost. Obviously, the cost of deleting T_i is equal to $|T_i|$ and the cost of repairing T_i is equal to $dist(T_i,D)$ (computed earlier). The cost of inserting a minimal subtree can be found using a simple algorithm (omitted here). A sequence of macro operations is a sequence repairing T if the resulting document is valid. The cost of a sequence of macro operations is the sum of the costs of its elements. A sequence of macro operations is equivalent on T to a sequence of atomic operations if their applications to T yield the same tree. Using the macro operations to repair trees is equivalent to atomic operations.

3.2 Restoration graph

Now, let $X_1,\ldots,X_n$ be the sequence of the labels of roots of $T_1,\ldots,T_n$ respectively. Suppose $E = D(X)$ defines the labels of children of the root and let $M_E = \langle \Sigma, S, q_0, \Delta, F\rangle$ be the NDFA recognizing $L(E)$ such that $|S| = O(|E|)$ [15].

To find an optimal sequence of macro operations repairing T, we construct a directed *restoration graph* U_T. The vertices of U_T are of the form q^i for $q \in S$ and $i \in \{0,\ldots,n\}$. The vertex q^i is referred as the *state* q *in the i-th column* of U_T and corresponds to the state q being reached by M_E after reading $X_1,\ldots,X_i$ processed earlier with some macro operations. The edges of the restoration graph correspond to the macro operations applied to the children of T:

– $q^{i-1} \xrightarrow{Del} q^i$ corresponds to deleting T_i and such an edge exists for any state $q \in S$ and any $i \in \{1,\ldots,n\}$,
– $q^{i-1} \xrightarrow{Rep} p^i$ corresponds to repairing T_i recursively and such an edge exists only if $\Delta(q,X_i,p)$,
– $q^i \xrightarrow{Ins_Y} p^i$ corresponds to inserting before T_i a minimal subtree labeled with Y and such an edge exists only if $\Delta(q,Y,p)$.

A *repairing path* in U_T is a path from q_0^0 to any accepting state in the last column of U_T.

Lemma 1. *For any sequence of macro operations v, v is a sequence repairing T w.r.t. D iff there exists a repairing path (possibly cyclic) in U_T labeled with the consecutive elements of v.*

If we assign to each edge the cost of the corresponding macro operation, the problem of finding a cheapest repairing sequence of macro operations is reduced to the problem finding a shortest path in U_T.

Theorem 1. *$dist(T,D)$ is equal to the minimum cost of a repairing path in U_T.*

Example 6. Figure 4 illustrates the construction of the restoration graph for the document `C(A(a),B(b),B())` and the DTD from Example 4. The automaton $M_{(\mathtt{A}\cdot\mathtt{B})^*}$ consists of two states q_0 and q_1; q_0 is both the starting and the only accepting state; $\Delta = \{(q_0,\mathtt{A},q_1),(q_1,\mathtt{B},q_0)\}$. The cheapest repairing paths are indicated with bold lines.

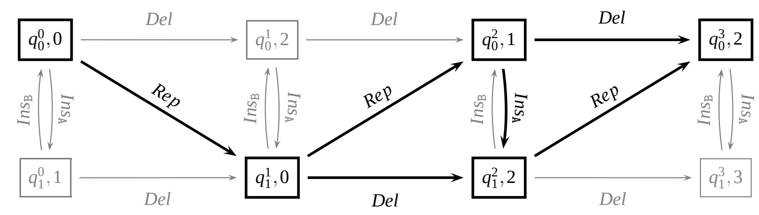


Fig. 4. Construction of the restoration graph.

In each vertex we additionally put the minimum cost of reaching that vertex from q_0^0. Note that the restoration graph represents 3 different repairs: (1) C(A(a), B(), A(), B()), obtained by inserting A(); (2) C(A(a), B()) obtained by deleting the second child; (3) C(A(a), B()) obtained by repairing the second child (removing the text node b) and removing the third child. We note that although isomorphic, the repairs (2) and (3) are not the same because the nodes labeled with B are obtained from different nodes in the original tree.

3.3 Trace graph

The *trace graph* U_T^* is the subgraph of U_T consisting of only the cheapest repairing paths. Note that if U_T has cycles, only arcs labeled with inserting macro operations can be involved in them. Since the costs of inserting operations are positive, U_T^* is a directed acyclic graph.

Repairs and paths in the trace graph Suppose now that we have constructed a trace graph in every node of T. Every repair can be characterized by selecting a path on each of the trace graphs. Similarly a choice of paths in each of the trace graphs yields a repair. We note that a choice of a path on the top-level trace graph of T may correspond to more than one repair (this happens when some subtree has more than one repair).

Complexity analysis First, note that for any vertex from the restoration graph U_T the incoming edges come from the same or the preceding column. Therefore, when computing the minimum cost of a path to a given vertex we need to consider at most $(|\Sigma|+1) \times |S| + 1$ values.

Moreover, we don't need to store the whole restoration graph in memory, but only its two consecutive columns. Also, note that we need the values $dist(T_i, D)$ and $|T_i|$ only when we compute the minimum cost for the i-th column of U_T, so there is no need for extra space to store these values. We assume that Σ is fixed and $|S|$ is bounded by $|D|$.

Theorem 2. *The distance between a document T and a DTD D can be computed in $O(|D|^2 \times |T|)$ time using $O(|D|^2 \times height(T))$ space.*

4 Valid query answers

In our paper we use the negation-free fragment of XPath 1.0 [26] restricted to its logical *core* (only element and text nodes, and only functions selecting the string value of nodes). Our approach, however, is applicable to a wider class of negation-free Regular XPath Queries [18]. We assume the standard semantics of XPath queries and by $QA^Q(T)$ we denote the answers to the query Q in the tree T.

We use an evaluation method that is geared towards the computation of valid answers. The basic notion is this of a *tree fact* (n,p,x) which states that an *object* x (a node, a label, or a string constant) is reachable from the node n with an XPath expression p.

We distinguish *basic* tree facts which use only $parent$::$*$, $following\text{-}sibling$::$*$, $name(.)$, and $text(.)$ path expressions. We note that basic tree facts capture all structural and textual information contained in XML documents. For the tree $T_1 = \mathtt{C}(\mathtt{A}(\mathtt{a}),\mathtt{B}(\mathtt{b}),\mathtt{B}())$ from Figure 3 examples of basic facts are: $(n_0, parent{::}{*}, n_3)$, $(n_3, parent{::}{*}, n_4)$, and $(n_4, text(), \mathtt{a})$. Other tree facts can be derived from the basic facts using simple Horn rules that follow the standard semantics of XPath. For example:

$$\begin{aligned}(x, descendant{::}{*}, y) &\longleftarrow (x, parent{::}{*}, y)\\ (x, descendant{::}{*}, y) &\longleftarrow (x, descendant{::}{*}, z) \land (z, parent{::}{*}, y)\\ (x, p_1/p_2, y) &\longleftarrow (x, p_1, z) \land (z, p_2, y)\end{aligned}$$

For instance, for the document T_1 we can derive $(n_0, descendant{::}{*}/text(.), \mathtt{a})$. Since we consider only positive queries, the used rules don't contain negation. Therefore, the derivation process, similarly to negation-free Datalog programs, is monotonic i.e., adding new (basic) facts does not invalidate facts derived previously.

Given a document T and a query Q we construct the set of all relevant tree facts B by adding to B all the basic facts of T. If adding a fact allows to derive new facts involving subexpressions of Q, these facts are also added to B. To find the answers to Q we simply select the facts that originate in the root of T and involve Q.

4.1 Validity-sensitive query evaluation

Definition 4 (Valid query answers). *Given a tree T, a query Q, and a DTD D, an object x is a* valid answer *to Q in T w.r.t D if x is an answer to Q in every repair of T w.r.t. D.*

Computing valid query answers We construct a bottom-up algorithm that for every node constructs the set of *certain* tree facts that hold in every repair of the subtree rooted in this node. The set of certain tree facts computed for the root node is used to obtain the valid answers to the query (similarly to standard answers).

We now fix the tree T, the DTD D, and the query Q, and assume that we have constructed the trace graph U_T^* for T as described in Section 3. We also assume that the sets of certain tree facts for the children of the root of T have been computed earlier.

Recall that the macro operation Ins_Y corresponds to inserting a minimum-size tree valid w.r.t. the DTD, whose root label is Y. Thus for every label Y our algorithm needs the set C_Y of (certain) tree facts present in every minimal valid tree with the root's label Y. This set can be constructed with a simple algorithm (omitted here).

4.2 Naive computation of valid answers

We start with a naive solution, which may need an exponential time for computation. Later on we present a modification which guarantees a polynomial execution time.

For each repairing path in U_T^* the algorithm constructs the set of certain tree facts present in every repair corresponding to this path. Assume now that $T = X(T_1,\dots,T_n)$, and the root nodes of $T, T_1,\dots,T_n$ are respectively $r, r_1,\dots,r_n$.

For a path $q_0^0 = v_0, v_1,\dots,v_m$ in U_T^* we compute the corresponding set C of certain facts in an incremental fashion (in every step we keep adding any facts that can be derived for subexpressions of the query Q):

1. for q_0^0 the set of certain facts consists of all the basic fact for the root node;
2. if C is the set corresponding to $v_0,\dots,v_{k-1}$, then the set C' corresponding to $v_0,\dots,v_k$ is obtained by one of the 3 following cases depending on the type of edge from v_{k-1} to v_k:
   - for $q^{i-1} \xrightarrow{Del} q^i$ no additional facts are added, i.e., $C' = C$;
   - for $q^{i-1} \xrightarrow{Rep} p^i$ we *append* the tree facts of T_i to C, i.e., we add to C certain facts of the tree T_i with the basic fact $(r, /*, r_i)$; if on the path $v_0,\dots,v_{k-1}$ other trees have been appended (with either *Rep* or Ins_Y instruction), then we also add the fact $(r', \mathit{following\text{-}sibling} :: *, r_i)$ where r' is the root node of the last appended tree;
   - $q^i \xrightarrow{Ins_Y} p^i$ is treated similarly to the previous case, but we append (a copy of) C_Y.

Naturally, the set of certain facts for T is the intersection of all sets corresponding to repairing paths in U_T^*. We implement this algorithm by computing for every v the collection $\mathbb{C}(v)$ of sets of tree facts corresponding to every path from q_0^0 to v.

Example 7. Recall the document $T_1 = \mathtt{C}(\mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{B}())$ and the trace graph from Figure 4 constructed for T_1 and DTD D_1 (Example 6). We consider the query $Q_1 = \mathit{descendant}::* /\mathit{text}(.)$ and we denote the operation of deriving tree facts involving subqueries of Q_1 with the superscript $(\cdot)^{Q_1}$. The collections for the trace graph U_T^* are constructed as follows:

$\mathbb{C}(q_0^0) = \{B_0\}$, where

$$B_0 = (\{(n_0, \mathit{name}(.), \mathtt{C}, n_0)\})^{Q_1}.$$

$\mathbb{C}(q_1^1) = \{B_1\}$, where

$$B_1 = (B_0 \cup C_1 \cup \{(n_0, \mathit{parent}::*, n_1)\})^{Q_1},$$

and C_1 is the set of certain facts for $\mathtt{A}(\mathtt{d})$

$$C_1 = (\{(n_1, \mathit{name}(.), \mathtt{A}), (n_1, \mathit{parent}::*, n_2), (n_2, \mathit{name}(.), \mathtt{PCDATA}), (n_2, \mathit{text}(.), \mathtt{d})\})^{Q_1}.$$

$\mathbb{C}(q_2^0) = \{B_2\}$, where

$$B_2 = (B_1 \cup C_2 \cup \{(n_0, \mathit{parent}::*, n_3), (n_1, \mathit{following\text{-}sibling}::*, n_3)\})^{Q_1},$$

and C_2 is the set of certain facts for the second child

$$C_2 = (\{, (n_3, \mathit{name}(.), \mathtt{B})\})^{Q_1}.$$

$\mathbb{C}(q_1^2) = \{B_1, B_3\}$, where

$$B_3 = (B_1 \cup C_{\mathtt{A}} \cup \{(n_0, \mathit{parent}{::}{*}, i_1), (n_3, \mathit{following\text{-}sibling}{::}{*}, i_1)\})^{Q_1},$$

where $C_{\mathtt{A}}$ is the set of certain facts for every valid tree with the root label A (i_1 is a new node)

$$C_{\mathtt{A}} = (\{(i_1, \mathit{name}(.), \mathtt{A})\})^{Q_1}.$$

$\mathbb{C}(q_0^3) = \{B_2, B_4, B_5\}$, where

$$B_4 = (B_3 \cup C_3 \cup \{(n_0, \mathit{parent}{::}{*}, n_5), (i_1, \mathit{following\text{-}sibling}{::}{*}, n_5)\})^{Q_1},$$

$$B_5 = (B_1 \cup C_3 \cup \{(n_0, \mathit{parent}{::}{*}, n_5), (n_1, \mathit{following\text{-}sibling}{::}{*}, n_5)\})^{Q_1},$$

where C_3 is the set of certain facts for the third child

$$C_3 = (\{(n_5, \mathit{name}(.), \mathtt{B})\})^{Q_1}.$$

In order to prevent an exponential explosion of the sizes of the consecutive collections, we use the following optimization of *eager intersection*:

> Let B_1 and B_2 be two sets from $\mathbb{C}(v)$ for some vertex v. Suppose that $v \longrightarrow v'$ and the edge is labeled with an operation that appends a tree (either *Rep* or *Ins*). Let B_1' and B_2' be the sets for v' obtained from B_1 and B_2 respectively. Instead of storing in $\mathbb{C}(v')$ both sets B_1' and B_2' we only store their intersection $B_1' \cap B_2'$.

In Example 7 this optimization give us: $\mathbb{C}(q_0^3) = \{B_2, B_{4,5}'\}$, where $B_{4,5}' = B_4 \cap B_5$.

With a simple induction over the column number we show that the number of sets of tree facts stored in a vertex in the i-th column is $O(i \times |S| \times |\Sigma|)$. We use the notion of *data complexity* [24] which allows to express the complexity of the algorithm in terms of the size of the document only (by assuming other input components to be fixed).

Theorem 3. *The data-complexity of computation of valid answers to negation-free core XPath queries is PTIME.*

We note that if we include the query into the input, the problem becomes co-NP-complete (we omit the proof). This shows that computing valid query answers is considerably more complex than computation of standard answers (whose combined complexity is known to be in PTIME [13]).

5 Experimental evaluation

In our experiments, we tested 2 algorithms: DIST computing $\mathit{dist}(D, T)$ and VQA computing valid answers. We compared these algorithms with an algorithm VALIDATE for validation of a document and an algorithm QA computing standard answers. All compared algorithms have been implemented using a common set of programming tools including: the parser, the representation of regular expressions and corresponding NDFA's, the representation for tree facts, and algorithms maintaining closure of the set of tree facts. For ease of implementation, we considered only a restricted class of *non-ascending path queries* which use only simple filter conditions (testing tag and text elements), do not use union, and involve only *child*, *descendant*, and *following-sibling* axises. We note that those queries are most commonly used in practice and the restrictions allow to construct algorithms that compute standard answers to such queries in time linear in the size of the document. This is also the complexity of the QA algorithm.

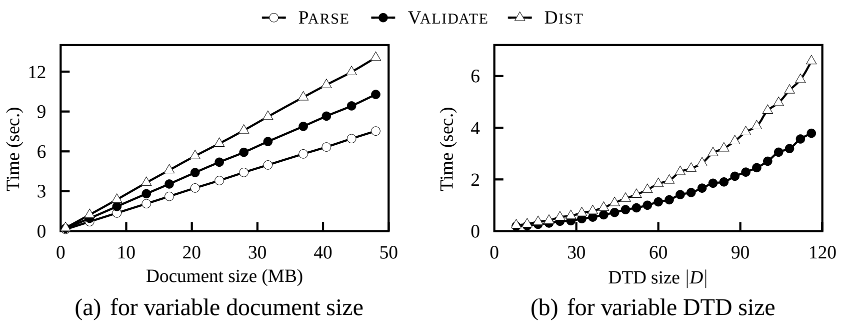
Data generation To observe the impact of the document size on the performance of algorithms, we randomly generated a valid document and we introduced the violations of validity to a document by randomly removing and inserting nodes. To measure the invalidity of a document T we use the *invalidity ratio* $dist(T,D)/|T|$. All the documents used for tests had a small height, 8-10.

For most of the experiments we used the DTD D_0 and the query Q_0 from Example 1. To measure the impact of the DTD size on the performance, we generated a family of DTDs D_n, $n \geq 0$: $D_n(\mathtt{A}) = (\ldots((\mathtt{PCDATA}+\mathtt{A}_1)\cdot\mathtt{A}_2+\mathtt{A}_3)\cdot\mathtt{A}_4+\ldots\mathtt{A}_n)^*$,$D_n(\mathtt{A}_i) = \mathtt{A}^*$. For those documents we used a simple query $//*/text(.)$.

Environment The system was implemented in Java 5.0 and tested on an Intel Pentium M 1000MHz machine running Windows XP with 512 MB RAM and 40 GB hard drive.

5.1 Experimental results

Results in Figure 5(a) and in Figure 5(b) confirm our analysis: edit distance between a document and a DTD can be computed in time linear in the document size and quadratic in the size of the DTD. If we take as the base line the time needed to parse the whole file (PARSE), then we observe that the overhead needed to perform computations is small. Because our approach to computing edit distance doesn't assume any particular properties of the automata used to construct the trace graph, Figure 5(b) allows us to make the following conjecture: Any techniques that optimize the automata to efficiently validate XML documents should also be applicable to the algorithm for computing the distance of XML documents to DTDs.



(a) for variable document size (b) for variable DTD size

Fig. 5. Edit distance computation time (0.1% invalidity ratio)

Figure 6(a) shows that for the DTD D_0 computing valid query answers is about 6 times longer than computing query answers with QA. Similarly to computing edit distance, computing valid answers involves constructing the restoration graph. This explains the quadratic dependency between the performance time and the size of DTD observed for VQA in Figure 6(b).

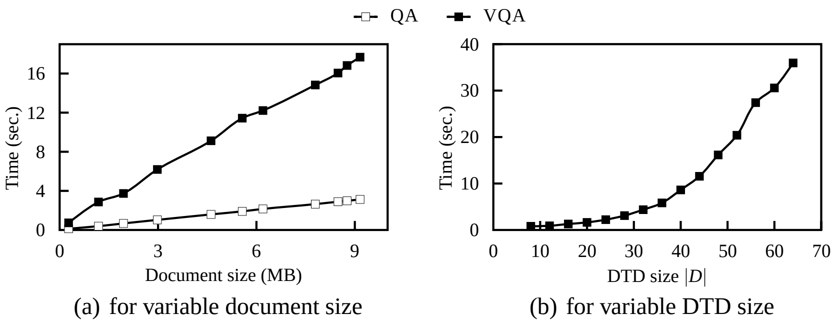


Fig. 6. Valid answers computation time (0.1% invalidity ratio).

6 Related work

Tree edit distance Tree edit distance is a generalization of the classical string edit distance. There are several different versions of the former notion varying with the semantics of tree operations [7]. In the most studied approach [23], the deleting operation also can be performed on a internal node, in which case the children are *promoted up*. Conversely, the inserting operation can *push down* a contiguous sequence of nodes. The implied notion of edit distance is not equivalent to ours (our notion is sometimes called *1-degree edit distance* [22]). In the area of data integration, insertions and deletions of internal document nodes could be used for the resolution of major structural discrepancies between documents. However, such operations require shifting nodes between levels and thus it is not clear if our approach can be adapted to that context. The notion of edit distance identical to ours has been used in papers dealing with the maintenance of XML integrity [1, 5, 6] and to measure structural similarity between XML documents [20]. [9] studies an extension of the basic tree edit framework with *moves*: a subtree can be shifted to a new location. In the context of validity-sensitive querying, extending our approach with move operations would allow to properly handle situations where invalidity of the document is caused by transposition of elements.

Almost every formulation of edit distance, including ours, allows to assign a non-unit cost to each operation.

Structural restoration A problem of correcting a *slightly invalid* document is considered in [9]. Under certain conditions, the proposed algorithm returns a valid document whose distance from the original one is guaranteed to be within a multiplicative constant of the minimum distance. The setting is different from ours: XML documents are encoded as binary trees, so performing editing operations on a encoded XML document may shift nodes between levels in terms of the original tree.

A notion equivalent to the distance of a document to a DTD (Definition 2) was used to construct error-correcting parsers for context-free languages [2].

Consistent query answers for XML [10] investigates querying XML documents that are valid but violate functional dependencies. Two repairing actions are considered: updating element values with a *null* value and marking nodes as unreliable. This choice of actions prevents from introducing invalidity in the document upon repairing it. Nodes with null values or marked as unreliable do not cause violations of functional dependencies but also are not returned in the answers to queries. Repairs are consistent instances with a minimal set of nodes affected by the repairing actions.

A set of operations similar to ours is considered for consistent querying of XML documents that violate functional dependencies in [11]. Depending on the operations used different notions of repairs are considered: *cleaning* repairs obtained only by deleting elements, *completing* repairs obtained by inserting nodes, and *general* repairs obtained by both operations.

[25] is another adaptation of consistent query answers to XML databases closely based on the framework of [4].

7 Conclusions and Future work

In this paper we investigated the problem of querying XML documents containing violations of validity of a local nature caused by missing or superfluous nodes. We proposed a framework which considers possible repairs of a given document obtained by applying a minimum number of operations that insert or delete nodes. We demonstrated algorithms for (a) measuring invalidity in terms of document-to-DTD distance, and (b) validity-sensitive querying based on the notion of valid query answer.

We envision several possible directions for future work. First, one can investigate if valid answers can be obtained using query rewriting [14]. Second, it is an open question if negation could be introduced into our framework. Third, it would be of significant interest to establish a complete picture of how the complexity of the computational problems considered in this paper (computing document-to-DTD distance, computing valid query answers) depends on the query language and the repertoire of the available tree operations (other operations include subtree swap, restricted subtree move). Finally, it would be interesting to find out to what extent our framework can be adapted to handle semantic inconsistencies in XML documents, for example violations of key dependencies.

References

1. S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. L. Wiener. Incremental Maintenance for Materialized Views over Semistructured Data. In *International Conference on Very Large Data Bases (VLDB)*, pages 38–49, 1998.

2. A. V. Aho and T. G. Peterson. A Minimum Distance Error-Correcting Parser for Context-Free Languages. *SIAM Journal on Computing*, 1(4):305–312, 1972.
3. S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *International Conference on Extending Database Technology (EDBT)*, pages 496–513, 2002.
4. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, 1999.
5. A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental Validation of XML Documents. *ACM Transactions on Database Systems (TODS)*, 29(4):710–751, December 2004.
6. M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. In *ACM Symposium on Principles of Database Systems (PODS)*, 2005.
7. P. Bille. Tree Edit Distance, Aligment and Inclusion. Technical Report TR-2003-23, The IT University of Copenhagen, 2003.
8. P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *ACM SIGMOD International Conference on Management of Data*, 2005.
9. U. Boobna and M. de Rougemont. Correctors for XML Data. In *International XML Database Symposium*, pages 97–111, 2004.
10. S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and Consistent Answers for XML Data with Functional Dependencies. In *International XML Database Symposium*, 2003.
11. S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Querying and Repairing Inconsistent XML Data. In *Web Information Systems Engineering (WISE)*, pages 175–188, 2005.
12. A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *ACM SIGMOD International Conference on Management of Data*, 2005.
13. G. Gottlob, C. Koch, and R. Pichler. XPath Processing in a Nutshell. *SIGMOD Record*, 32(2):21–27, 2003.
14. G. Grahne and A. Thomo. Query Answering and Containment for Regular Path Queries under Distortions. In *Foundations of Information and Knowledge Systems (FOIKS)*, 2004.
15. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2nd edition, 2001.
16. A. Klarlund, T. Schwentick, and D. Suciu. XML: Model, Schemas, Types, Logics and Queries. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.
17. W. L. Low, W. H. Tok, M. Lee, and T. W. Ling. Data Cleaning and XML: The DBLP Experience. In *International Conference on Data Engineering (ICDE)*, page 269, 2002.
18. M. Marx. Conditional XPath, the First Order Complete XPath Dialect. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 13–22, 2004.
19. F. Neven. Automata, Logic, and XML. In *Workshop on Computer Science Logic (CSL)*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.
20. A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *Workshop on the Web and Databases (WebDB)*, pages 61–66, 2002.
21. P. Polyzotis, M. N. Garofalakis, and Y. E. Ioannidis. Approximate XML Query Answers. In *ACM SIGMOD International Conference on Management of Data*, pages 263–274, 2004.
22. S. M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters*, 1977.
23. D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In A. Apostolico and Z. Galil, editors, *Pattern Matching in Strings, Trees, and Arrays*, pages 341–371. Oxford University Press, 1997.
24. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.
25. S. Vllalobos. Consistent Answers to Queries Posed to Inconsistent XML Databases. Master's thesis, Catholic University of Chile (PUC), 2003. In Spanish.
26. W3C. XML path language (XPath 1.0), 1999.