XML Query Optimisation: Specify your Selectivity

Sven Hartmann, Sebastian Link
Information Science Research Centre
Massey University, New Zealand
EMail: [s.hartmann,s.link]@massey.ac.nz

Abstract

The problem of efficiently evaluating XPath and XQuery queries has become increasingly significant since more and more XML data is stored in its native form.
We propose a novel optimisation technique for XML queries that is based on the semantic properties exhibited by XML data. In sharp contrast to previous studies on selectivity estimation we propose to specify bounds on the number of element nodes in an XML tree that form the root of isomorphic subtrees. It turns out that efficient reasoning about these constraints provides effective means to predict the number of XPath and XQuery query answers, to predict the number of updates using the XQuery update facility, to predict the number of en(de)cryptions using XML encryption, and to optimise XML queries.

Keywords: XML, Query, Optimisation, Selectivity, Constraints

1 Introduction

The eXtensible Markup Language (XML) has evolved to the de-facto standard for data exchange on the World Wide Web. This development has also resulted in a rapid increase of XML data that has to be stored in its native format. It is therefore a big challenge for the database community to design query languages and storage methods in order to retrieve data from vast amounts of XML data efficiently. In this context, many query languages for XML such as Lorel, Quilt, XQL, XML-QL, XPath and XQuery have been proposed. The two most popular among these query languages are XPath and its superset, XQuery.
The XQuery language is still a young standard, and many optimisation techniques remain unexplored. Since more and more XML documents are stored and exchanged, the efforts of evaluating XPath and XQuery queries efficiently increase as well. In the following we will briefly discuss some previous work on XQuery optimisation in order to pinpoint the differences to our optimisation strategies.

Related Work. The literature on XML query optimisation is very vast. An early article on this subject is [10], and a recent survey that includes many more references is [7]. Schemata force XML documents to conform to a specific structure, and can therefore be viewed as constraints. A few papers deal with query optimisation in the presence of schemata [3, 12]. A general approach towards query rewriting under constraints is the chase and backchase [4], but the emphasis of this technique is on data integration. There is also a large amount of literature on selectivity estimation, and we only mention [1, 11, 13]. While this line of research estimates the number of nodes selected by a query, we propose to utilise XML constraints to infer such bounds. Our novel class of constraints subsume XML keys [2, 6], and have therefore a greater potential for query optimisation. In fact, the exploration of XML keys for XML query optimisation is proposed as future work in [2], and has been started in [5]. Our constraints also generalise cardinality constraints from Entity-Relationship models [8, 9].

Contributions. We introduce a novel method for query optimisation that is based on the specification of constraints and takes therefore advantage of semantic properties exhibited by XML data. Efficient reasoning about these constraints can help to determine the selectivity of XML queries effectively. This is in sharp contrast to research on selectivity estimation since our optimisation techniques result always in a better performance, if they are applicable.

Organisation. We summarise the underlying XML tree model, the notion of value equality and relevant path languages in Section 2. Then we introduce numerical constraints in Section 3 and illustrate their relevance by numerous examples. We will describe how efficient reasoning about numerical constraints results in effective means to determine bounds on the number of query answers, updates and XML en(de)cryptions in Section 4. Moreover, we will illustrate in Section 5 how XPath and XQuery queries can be optimised in the presence of numerical constraints. Finally, we use Section 6 to list several results on the intractability and tractability of various subclasses of numerical constraints. We conclude in Section 7.

1529-4188/07 $25.00 © 2007 IEEE
DOI 10.1109/DEXA.2007.19

IEEE computer society

2 The XML tree model

XML documents can be modelled as node-labelled trees. We assume that there is a countably infinite set **E** denoting element tags, a countably infinite set **A** denoting attribute names, and a singleton $\{S\}$ denoting text (PCDATA). We further assume that these sets are pairwise disjoint, and put $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \{S\}$. We refer to the elements of $\mathcal{L}$ as *labels*. An *XML data tree* is a 6-tuple $T = (V, lab, ele, att, val, r)$ where V denotes a set of nodes, and *lab* is a mapping $V \to \mathcal{L}$ assigning a label to every node in V. A node $v \in V$ is called an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) = S$. Moreover, *ele* and *att* are partial mappings defining the edge relation of T: for any node $v \in V$, if v is an element node, then $ele(v)$ is a list of element and text nodes in V and $att(v)$ is a set of attribute nodes in V. If v is an attribute or text node then $ele(v)$ and $att(v)$ are undefined. *val* is a partial mapping assigning a string to each attribute and text node: for each node $v \in V$, $val(v)$ is a string if v is an attribute or text node, while $val(v)$ is undefined otherwise. Finally, r is the unique and distinguished root node. For example, Figure 1 illustrates an XML data tree in which data about a company's projects is stored.

An important notion is value equality for pairs of nodes in an XML tree. Informally, two nodes u and v of an XML tree T are value equal if they have the same label and, in addition, either they have the same string value if they are text or attribute nodes, or their children are pairwise value equal if they are element nodes. For instance, in Figure 1 nodes v_{37} and v_{53} are value equal while nodes v_5 and v_{21} are not value equal.

In order to define our constraints we need a path language that is expressive enough to be practical, yet sufficiently simple to be reasoned about efficiently. This is the case for the path languages PL_s and PL [2].

Path Language	Syntax
PL_s	$P ::= \varepsilon \mid \ell.P$
PL	$Q ::= \varepsilon \mid \ell \mid Q.Q \mid \_^*$

Table 1. The path languages PL_s and PL.

A *path expression* is a (possibly empty) finite list of symbols. In this paper, a *simple path expression* is a path expression that consists of labels from $\mathcal{L}$. We use the languages PL_s and PL to describe path expressions. Both languages are fragments of regular expressions. PL_s *expressions* and PL *expressions* are defined by the grammars in Table 1. Herein, ε denotes the empty path expression, "." denotes the concatenation of two path expressions, and ℓ denotes any element of $\mathcal{L}$. The language PL is a generalisation of PL_s that allows the distinguished symbol "$\_^*$" to occur. We call $\_^*$ the *don't care* symbol. It serves as a combination of the wildcard "$\_$" and the Kleene star "*". Note that every PL_s expression is a PL expression, too.

When replacing all $\_^*$ in a PL expression Q by simple path expressions, we obtain a simple path expression P and write $P \in Q$. Thus, a PL expression Q gives rise to a regular language of simple path expressions $P \in Q$. We use path expressions to describe sets of paths in an XML tree T. Recall that each attribute or text node is a leaf in T. Therefore, a path expression is said to be *valid* if it does not contain a label $\ell \in \mathbf{A}$ or $\ell = S$ in a position other than the last one. In the sequel, we use a valid PL expression to represent only valid simple path expressions.

A path p is a sequence of pairwise distinct nodes $v_0, \ldots, v_m$ where (v_{i-1}, v_i) is an edge for $i = 1, \ldots, m$. We call p a path from v_0 to v_m, and say that v_m is *reachable* from v_0 following the path p. The path p gives rise to a valid simple path expression $lab(v_1).\cdots.lab(v_m)$, which we denote by $lab(p)$. Let P be a simple path expression, and Q a PL expression. A path p is called a P*-path* if $lab(p) = P$, and a Q*-path* if $lab(p) \in Q$. If p is a path from v to w, then w is said to be reachable from v *following* a P-path or Q-path, respectively. For instance, in Figure 1 the node v_{31} is reachable from the node v_{19} by following a $\_^*$.mng-path.

For a node v of an XML tree T, let $v[\![Q]\!]$ denote the set of nodes in T that are reachable from v by following the PL expression Q. We shall use $[\![Q]\!]$ as an abbreviation for $r[\![Q]\!]$ where r is the root of T.

For nodes v and v' of T, the *value intersection* of $v[\![Q]\!]$ and $v'[\![Q]\!]$ is given by $v[\![Q]\!] \cap_v v'[\![Q]\!] = \{(w, w') \mid w \in v[\![Q]\!], w' \in v'[\![Q]\!], w =_v w'\}$ [2]. That is, $v[\![Q]\!] \cap_v v'[\![Q]\!]$ consists of all those node pairs in T that are value equal and are reachable from v and v', respectively, by following Q-paths. For example, in Figure 1 we have $v_3[\![\_^*.\text{mem}]\!] = \{v_9, v_{11}, v_{17}\}$, $v_{43}[\![\_^*.\text{mem}]\!] = \{v_{49}, v_{51}, v_{57}\}$ and $v_3[\![\_^*.\text{mem}]\!] \cap_v v_{43}[\![\_^*.\text{mem}]\!] = \{(v_9, v_{49}), (v_{11}, v_{57}), (v_{17}, v_{51})\}$.

3 Numerical constraints

XML data typically conforms to various kinds of constraints which can be specified by the data administrator in order to avoid inconsistencies in and processing difficulties with the database. If the data is known to satisfy the constraints that have been specified, then the knowledge about the constraints can be explored in various ways, for instance to optimise queries. For instance, the data shown in Figure 1 conforms to the following business rule: In every month, every employee can manage at most 2 different projects. However, the data violates the business rule that in every month every manager must manage exactly 2 different projects (e.g. *Taz* only manages the project with

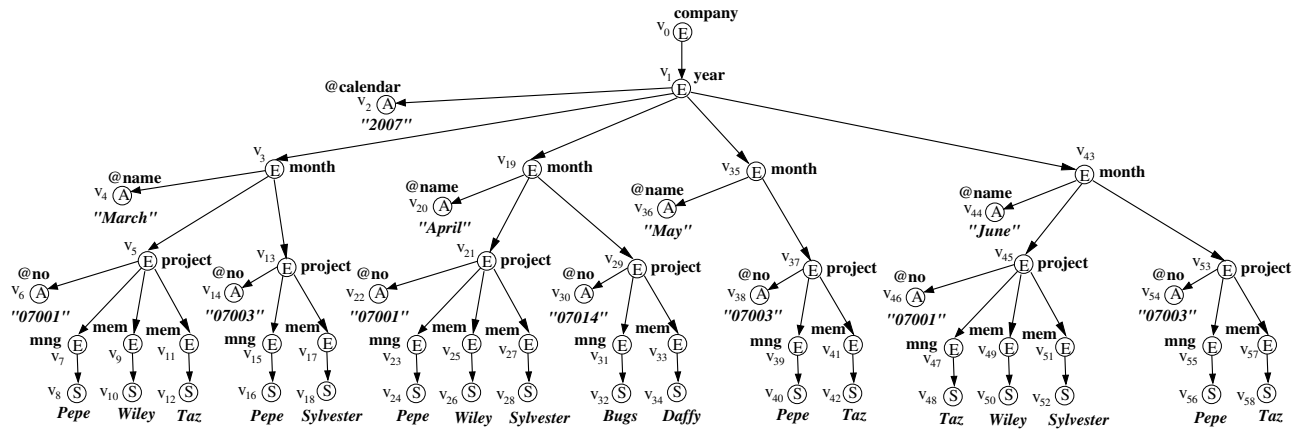

Figure 1. An XML data tree fragment

project number *07001* in the month of *June*). We will now formalise these constaints.

Let $\mathbb{N}$ denote the positive integers, let $\bar{\mathbb{N}}$ denote the positive integers together with ∞ and let $\sharp S$ denote the cardinality of a finite set S, i.e., the number of its elements. A *numerical constraint* φ for XML is an expression $\mathit{card}(Q, (Q', \{Q_1, \ldots, Q_k\})) = (\min, \max)$ where $Q, Q', Q_1, \ldots, Q_k$ are *PL* expressions such that $Q.Q'$ is a valid path expression if $k = 0$, and $Q.Q'.Q_i$ are valid path expressions for all $i = 1, \ldots, k$ if $k > 0$, where k is a non-negative integer, and where $\min \in \mathbb{N}$ and $\max \in \bar{\mathbb{N}}$ with $\min \leq \max$. Herein, Q is called the *context path*, Q' is called the *target path*, $Q_1, \ldots, Q_k$ are called *key paths*, $\min$ is called the *lower bound*, and $\max$ the *upper bound* of φ. If $Q = \epsilon$, we call φ *absolute*; otherwise φ is called *relative*. We use $\mathcal{N}$ to denote the class of all numerical constraints. An XML tree T *satisfies* φ, denoted by $T \models \varphi$, if and only if for all $q \in [\![Q]\!]$, for all $q' \in q[\![Q']\!]$ such that for all $x_1, \ldots, x_k$ with $x_i \in q'[\![Q_i]\!]$ for $i = 1, \ldots, k$, the following holds:

$$\min \leq \sharp\{q'' \in q[\![Q']\!] \mid \exists y_1, \ldots, y_k \text{ such that } y_i \in q''[\![Q_i]\!] \text{ and } x_i =_v y_i \text{ for } i = 1, \ldots, k\} \leq \max .$$

Notice that our constraints subsume the class of XML keys [2]. In fact, the key $(Q, (Q', \{Q_1, \ldots, Q_k\}))$ is satisfied by an XML tree T precisely if T satisfies the numerical constraint $\mathit{card}(Q, (Q', \{Q_1, \ldots, Q_k\})) = (1, 1)$. In the following we will briefly illustrate numerical constraints by various examples.

The first constraint says that *year*-nodes can be identified in the entire XML tree by the values on their *calendar*-child: $\mathit{card}(\epsilon, (\text{year}, \{\text{calendar}\})) = (1, 1)$. The constraint $\mathit{card}(\text{year}, (\text{month}, \emptyset)) = (1, 12)$ tells us that years for which projects are stored may store projects for every month of the year. In fact, in every subtree rooted at some *year*-node every *month*-node can be identified by the value on their *name*-child: $\mathit{card}(\text{year}, (\text{month}, \{\text{name}\})) = (1, 1)$. Moreover, in every *month* every *project* can be identified by its project number *no*, i.e., $\mathit{card}(\_^*.\text{month}, (\text{project}, \{\text{no}\})) = (1, 1)$. In addition the company implements the following business rules for project managers and members. Every project has precisely one manager: $\mathit{card}(\_^*.\text{project}, (\text{mng}, \emptyset)) = (1, 1)$, and every project has between 2 and 5 additional members (note that this constraint is violated by the data fragment in Figure 1): $\mathit{card}(\_^*.\text{project}, (\text{mem}, \emptyset)) = (2, 5)$. Projects have a duration of up to 6 months: $\mathit{card}(\epsilon, (\_^*.\text{month}, \{\text{project.no}\})) = (1, 6)$. In every month every employee can manage at most 2 different projects, i.e., $\mathit{card}(\_^*.\text{month}, (\text{project}, \{\text{mng}\})) = (1, 2)$, and every employee can participate in up to 3 different projects: $\mathit{card}(\_^*.\text{month}, (\text{project}, \{\text{mem}\}) = (1, 3)$.

Having specified some constraints an XML data tree is considered *legal* if it satisfies all of these constraints. However, constraints are not always satisfied independently from each other. In fact, there are some constraints which are implied by others. In order to fully explore the properties of XML data that conform to a set of constraints we must have efficient means to decide the implication of these constraints. Let $\Sigma \cup \{\varphi\}$ be a finite set of constraints in $\mathcal{N}$. We say that Σ *(finitely) implies* φ, denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies Σ also satisfies φ. The *(finite) implication problem for* $\mathcal{N}$ is to decide, given any finite set of constraints $\Sigma \cup \{\varphi\}$ in $\mathcal{N}$, whether $\Sigma \models_{(f)} \varphi$.

For example, the constraints that we have specified previously imply the following constraint: in every year each employee can manage up to 24 different projects. However, if the senior management of the company feels that 24 projects are too many to manage for a single employee, an additional constraint can be specified say $\mathit{card}(\text{year}, (\_^*.\text{project}, \{\text{mng}\})) = (1, 10)$.

4 Infering the number of Query Answers, Updates and Encryptions

We will demonstrate in this section how reasoning about numerical constraints enables the database management systems to infer lower and upper bounds on the number of query answers without actually querying the database itself. Such a tool does not only save potentially huge computation costs and resources but also allows the system to inform users about the costs that they will be charged for this service in case they decide to use it. In sharp contrast to previous work on estimating the number of query answers our technique is not based on any heuristics but provides bounds which will definitely hold. Consider for instance the following XQuery query

for $p **in**
doc("projects.xml")/year[@calendar="2007"]//project
where $p/mng="Pepe"
return ⟨project⟩{$p/no}⟨/project⟩

which returns the numbers of all those projects in 2007 which were managed by *Pepe*. In case the constraint $\varphi = card(\text{year}, (\_^*.\text{project}, \{\text{mng}\})) = (1, 10)$ has indeed been specified in addition to the other constraints above, it is easy to see that there can be at most ten *no*-nodes returned by this query. If φ has not been specified, then the system will be able to infer that the constraint $card(\text{year}, (\_^*.\text{project}, \{\text{mng}\})) = (1, 24)$ is implied by the other constraints above. Consequently, the upper bound on the number of *no*-nodes returned by this query is 24.

In the same way one is able to make predictions for the number of updates, using for instance the XQuery update facility. For example, the XQuery query

for $m **in** doc("projects.xml")/year[@calendar="2007"]/
month[@name="May" or @name="June"]//
mem[text()="Sylvester"]
return do replace value of $m/text() **with** "Tweety"

will update the text content *Sylvester* of *mem*-nodes by *Tweety* in all project memberships of this member in May and June of 2007. Under the very reasonable assumption that the same employee is not stored more than once within each *project*-subtree the maximal number of updates this query causes is 6.

When XML data is exchanged over the Web it is very common that sensitive information is encrypted, e.g. by XML encryption. In order to evaluate queries on encrypted XML data it may become necessary to decrypt certain data element nodes in order to return the relevant information in the answer. If we recall the example of the XQuery query above and assume that project number attributes have been encrypted, then a database management system capable of inferring that $\text{card}(\text{year}, (\_^*.\text{project}, \{\text{mng}\})) = (1, 24)$ is implied by the constraints specified can also predict that the number of necessary decryptions to answer this query is at most 24, and thus also predict the time necessary to deliver the information requested.

5 Query Optimisation

A considerable amount of research has been directed towards making XML query processing efficient. This includes many different query optimisation strategies, in particular on selectivity estimation.

In this section we propose to optimise XML queries utilising the semantic properties exhibited by the underlying XML data. More precisely, we take advantage of the numerical constraints specified by the data administrator to rewrite or simplify queries.

In order to demonstrate the potential of this technique we look at a few examples. The following query selects numbers of those projects from 2007 that feature the employee *Taz* as a member and are managed by the employee *Pepe*.

for $p **in**
doc("projects.xml")/year[@calendar="2007"]//project
where $p/mem/text()="Taz" **and** $p/mng/text()="Pepe"
return ⟨project⟩{$p/no}⟨/project⟩

While the **and** operator is commutative with respect to the query result the order of its inputs does have an impact on the query processing time. Due to the numerical constraints specified we can conclude that the selectivity of *project*-nodes based on the *mng*-subnodes is smaller than the selectivity based on the *mem*-subnodes. If queries are evaluated from left to right, then the query above should be rewritten into the following query

for $p **in**
doc("projects.xml")/year[@calendar="2007"]//project
where $p/mng/text()="Pepe" **and** $p/mem/text()="Taz"
return ⟨project⟩{$p/no}⟨/project⟩

which performs better due to the smaller selectivity of *project*-nodes based on *mng*- rather than *mem*-subnodes.

The following XQuery query retrieves *no*-children of those projects which only feature members that participated in at most five different projects in April 2007.

for $p **in**
doc("projects.xml")/year[@calendar="2007"]//project
where every $m **in** $p/mem **satisfies**
count(doc("projects.xml")/year[@calendar="2007"]//
month[@name="April"]/project[mem=$m])≤ 5
return ⟨project⟩{$p/no}⟨/project⟩

Having specified the constraint that in every month every employee can participate in at most three different projects the complex condition is satisfied by the underlying XML

tree whenever it already satisfies this constraint. Therefore, the query simply needs to retrieve the numbers of all projects from April 2007.

for \$p **in**
doc("projects.xml")/year[@calendar="2007"]//project
return ⟨project⟩{\$p/no}⟨/project⟩

There is the opportunity of a hybrid approach to query optimisation: one may utilize numerical constraints, but once their applicability has been exhausted we can use the derived information to make more accurate estimates.

6 Intractability and Tractability Results

Deciding implication for numerical constraints is not that easy, in general. In fact, reasoning is likely to be computationally intractable already for very restricted classes of numerical constraints. We call a numerical constraint $\text{card}(P, P', \{P_1, \ldots, P_k\}) = (\min, \max)$ *simple* if $P, P', P_1, \ldots, P_k$ are all simple path expressions in PL_s.

Theorem 1 *The finite implication problem for the class of all simple absolute numerical constraints with a non-empty set of key paths is $coNP$-hard.*

The previous result suggests that computational intractability may result from the specification of both lower and upper bounds. The next result indicates that an empty set of key paths may cause computational intractability.

Theorem 2 *The finite implication problem for the class of all simple absolute numerical constraints where the lower bound is fixed to 1 is $coNP$-hard.*

The next theorem suggests another source of computational intractability: the permission to have arbitrary path expressions in both target- and key paths.

Theorem 3 *The finite implication problem for the class of all absolute numerical constraints that have a non-empty set of key paths and where the lower bound is fixed to 1 is $coNP$-hard.*

The previous results motivate the study of a large subclass of numerical constraints that turns out to be computationally tractable. A *numerical key* for XML is a numerical constraint $\textit{card}(Q, (Q', \{P_1, \ldots, P_k\})) = (1, \max)$ where k is a positive integer and $P_1, \ldots, P_k$ are simple path expressions in PL_s.

Theorem 4 *The implication and finite implication problems for numerical keys coincide.*

Our final result shows that numerical keys form a large tractable subclass of numerical constraints.

Theorem 5 *The implication problem for the class of all numerical keys is finitely axiomatisable, and can be decided in time quadratic in the size of the constraints given.*

Efficient query optimisation is thus not limited to those numerical keys explicitely specified by the data administrator but subsumes also all those numerical keys that are implied. This makes query optimisation much more effective.

7 Future Work

We conclude the article by listing some open problem that warrant future research. First, we would like to devise a tool that automatically optimises XML queries under the presence of numerical constraints. The results of Section 6 direct our main focus on numerical keys. Furthermore, we want to study the consistency problem of numerical constraints, i.e., devise algorithms that decide whether an XML tree is consistent with respect to a set of numerical constraints, and investigate their computational behaviour.

References

[1] A. Aboulnaga, A. Alameldeen, and J. Naughton. Estimating the selectivity of XML path expressions for internet scale applications. In *VLDB*, pages 591–600, 2001.

[2] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.

[3] D. Che, K. Aberer, and M. Tamer Özsu. Query optimization in XML structured-document databases. *VLDB Journal*, 15(3):263–289, 2006.

[4] A. Deutsch, L. Popa, and V. Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[5] M. Essid, O. Boucelma, and S. Bressan. Answering queries in the presence of XML keys. In *DEXA Workshops*, pages 476–481, 2006.

[6] S. Hartmann and S. Link. Unlocking keys for XML trees. In *ICDT*, number 4353 in LNCS, pages 104–118. Springer, 2007.

[7] S. Haw and G. Radha Kishna Rao. Query optimization techniques for XML databases. *International Journal of Information Technology*, 2(2):97–104, 2005.

[8] M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Inf. Syst.*, 15(4):453–461, 1990.

[9] S. Liddle, D. Embley, and S. Woodfield. Cardinality constraints in semantic data models. *Data & Knowledge Engineering*, 11:235–270, 1993.

[10] J. McHugh and J. Widom. Query optimization for XML. In *VLDB*, pages 315–326, 1999.

[11] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Selectivity estimation for XML twigs. In *ICDE*, pages 264–275, 2004.

[12] H. Su, E. Rundensteiner, and M. Mani. Semantic query optimization for XQuery over XML streams. In *VLDB*, pages 277–288, 2005.

[13] Y. Wu, J. Patel, and H. Jagadish. Estimating answer sizes for XML queries. In *EDBT*, pages 590–608, 2002.