Catalog Integration Made Easy*

Pedro José Marrón, Georg Lausen and Martin Weber
Universität Freiburg, Institut für Informatik
Georges-Koehler-Allee, Geb. 51
79110 Freiburg, Germany
{pjmarron, lausen, weber}@informatik.uni-freiburg.de

Abstract: In this paper, we study adaptive evaluation techniques for querying XML-based electronic catalogs, and show, by means of experiments performed on real-world catalogs, that our approach can be used to integrate them with virtually zero effort at start-up time, and a small constant factor needed to perform the adaptive evaluation for all subsequent queries. We reach the conclusion that, from a strictly technical perspective, the classic role of the global catalog can be assumed in an ad-hoc manner by any catalog that forms part of a collaborative federation of XML-based catalogs, and implements our adaptive query algorithms, independently of the storage model used to access its contents.

Keywords: E-Commerce, E-Catalog, Catalog integration, XML, XPath

1 Introduction

The proliferation of XML-based catalog research [FLM98, Jhi00, Kel97], has made it easier for companies and suppliers to integrate their offers into common catalogs that allows them to reach customers in an easier way. Despite improvements in that area, the problems found in the classical view approach to catalog integration [SH01] have not been solved. The fact that the integration of different catalogs implies the generation, maintenance and adaptation of a global catalog that forwards the appropriate queries to the corresponding local catalogs for evaluation, requires us to perform query rewriting in order to provide the user with an answer.

In this paper, we evaluate, extend and improve on the techniques developed in [LM02] for querying XML-based electronic catalogs, that eliminates the need to perform an explicit query rewriting. Our methodology uses an adaptive query evaluation strategy that allows us to perform the same query on all local catalogs and still obtain accurate answers.

The rest of this paper is organized as follows. Section 2 explains the gist of our algorithms, introducing the types of problems found, but not addressed, in our previous work. In section 3, we refine our approach in order to improve on the deficiencies of the basic

* A short version of this paper has been accepted as a poster contribution for ICDE 2003 to be held in March 2003 in Bangalore, India.

system, leaving for section 4 the experimental evaluation of both the basic and refined systems. Finally, section 5 deals with related work in this area, and section 6 concludes this paper.

2 Adaptive Query Evaluation

Our approach to querying XML-based catalogs adaptively relies on the following three pillars: (1) Since, according to the XPath query model, an XPath query is evaluated by decomposing it into smaller, independent pieces, each of them can be transformed on-the-fly and adapt to the current catalog structure; (2) the use of a flexible fitness function allows us to discriminate more accurate solutions from others; and (3) the hierarchical nature of the conceptual organization of product catalogs makes it possible to eliminate or introduce concept categories at certain levels without affecting the outcome of the query.

Let us first revise the classic evaluation model found in the XPath standard [CD99].

2.1 XPath Evaluation Model

A formal characterization of this model, originally proposed in [ML01], can be summarized as follows:

Definition (XPath Query) An XPath Query Q_X is defined as $Q_X = q_0/q_1/\ldots/q_n$, where q_i is an XPath subquery defined below, and '/', the XPath subquery separator. □

Definition (XPath Subquery) An XPath Subquery q_i is a 3-tuple $q_i = (C_i, w_i, C_{i+1})$, where: C_i is a set of XML nodes that determine the input context; w_i is the Path Expression to be applied to each node of the input context (defined below); and C_{i+1} is a set of XML nodes resulting from the application of the path expression w_i onto the input context C_i. C_{i+1} is also called the output context.

An XPath subquery is also called a *location step* in the terminology used by the World Wide Web Consortium [W3C]. □

Definition (XPath Path Expression) A Path Expression w_i is a 3-tuple $w_i = a_i :: e_i[c_i]$ such that: a_i is an axis along which the navigation of the path expression takes place[1]; e_i is a node expression that tests either the name of the node or its content type; and c_i is a boolean expression of conditional predicates that must be fulfilled by all nodes along the path. □

[1] Only the `child` and `parent` axis are considered in our adaptive evaluation algorithm

2.2 Adaptive Query Strategy

The adaptive query strategy involves two main steps: a preprocessing step, where, if needed, each individual concept found in the query is translated based on the elements present in the local catalog; and the actual processing of the query.

Each local catalog is expected to provide a mapping from the concepts found in the global catalog to its own local representation, so that discrepancies that might appear as a result of using synonyms, or other languages for the same concepts are easily solved. For the purposes of our algorithms, we assume that this mapping is provided by translating the set of concepts transmitted as part of the query without paying attention to either structural or semantical relationships between the original query and the local catalog contents.

After the preprocessing step has completed, our algorithm performs the following steps: (1) Application of each adaptive transformation on the input context of each subquery; (2) evaluation of the fitness function at each step to provide a node ranking used to discriminate one set of solutions over another.

Definition (Subquery Transformations) The three possible transformations performed on a subquery q_i by the first step of our algorithm are: *No transformation (n):* Where the query is evaluated as it was originally specified by the user; *subquery generalization (g):* Where the axis of a particular subquery is augmented according to the following rules: if the original axis (a_i) is `child`, the augmented axis (a'_i) becomes `descendent`. Similarly, if a_i is `parent`, a'_i becomes `ancestor`; *subquery elimination (e):* A subquery is eliminated from the original XPath query if its result set (output context) is empty, allowing for the further evaluation of the following queries as if it had never been on the original query. □

In order to determine the correct sequence of transformations to apply at each subquery, given a subquery q_i with input context C_i, we evaluate it with respect to each subquery transformation, so that from each input context C_i, we generate three output contexts: C^n_{i+1}, C^g_{i+1} and C^e_{i+1} that correspond, respectively, to the application of the *no transformation (n)*, *subquery generalization (g)* and *subquery elimination (e)* transformations defined above. Each context $C_{i+1} = C^n_{i+1} \cup C^g_{i+1} \cup C^e_{i+1}$ is formed by the union of the result contexts obtained by the application of each one of the three transformations.

In order to distinguish the nodes that compose the optimal solution to the query we now define a fitness function that assigns a metric on the nodes based on the type of transformation used to generate them. Let us now define the form of the (global) fitness function that was originally proposed in [LM02], and that will be used later in comparison to other variants for the experiments of section 4.

Definition ((Global) Fitness function) Let q_i be the current subquery to be evaluated, and C_i its associated input context. Each node $n \in C_i$ is augmented to have a value v_n resulting from the evaluation of the previous subqueries $q_0 \ldots q_{i-1}$.

Then, the fitness function assigns a value v_m to each new node in $m \in C_{i+1}$ generated by a node $n \in C_i$ as follows: if $m \in C^n_{i+1}$, then $v_m = b^2 + v_n$; if $m \in C^g_{i+1}$, then $v_m = b + v_n$; and if $m \in C^e_{i+1}$, then $v_m = 1 + v_n$. b is a positive integer that represents the base of the fitness function.

If a node n has been generated by the application of more than one strategy, the final context C_{i+1} only contains the instance of n with the biggest value v_n; finally $v_{root} = 1$.□

As can be deduced from the definition of the fitness function, nodes generated as a result of the evaluation of a subquery in its original state have a higher fitness value than those generated by the generalization of the subquery, or those obtained by its elimination. The reason behind this choice is to favor the original query over transformation operations that might abstract away the intention of the user.

2.3 Formal Definition and Analysis

The distinction between the three types of transformations implies that the fitness function we just described provides an efficient way to traverse the partial order formed by the evaluation process of a particular query.

In fact, since at each step of the query evaluation we keep the results generated by each transformation, the contents of the last context at the end of the computation (as defined in subsection 2.2), correspond to the set of nodes found performing a series of *no transformations* ($nn \ldots n$), another set that corresponds to always performing *elimination* ($ee \ldots e$), and many other possibilities in between.

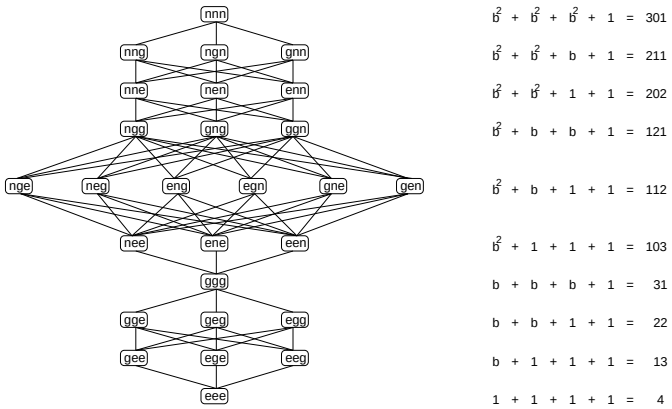

Figure 1: Partial Order for $q_l = 3$ and $t = 3$

Figure 1 shows the partial order generated for a query of length three ($q_l = 3$), by the application of the global fitness function defined in the previous section for $b = 10$, assuming that the number of allowed transformations is three ($t = 3$). As it is shown on the right-hand side of Figure 1, each level is assigned a different value by the fitness function, so that levels closer to nnn have higher values than other levels further down in the ordering.

Of course, it is not clear whether or not it is correct to say that $nne > ngg$, or that $nee > ggg$, as determined by our fitness function. In any case, given such a partial order, we can reorganize each level at our convenience and extract another fitness function that maps the new structure, as long as the new partial order does not contain any contradictions in the inequalities it defines. For example, assuming that the fitness function simply adds

the fitness values of each transformation, the ordering of Figure 1 determines that $3 \cdot n > 2 \cdot n + g$, $2 \cdot n + g > 2 \cdot n + e$, and so on, which leaves us with the following inequalities: $n > g$, $g > e$, $n - 2 \cdot g + e > 0$ and $n - 3 \cdot g + 2 \cdot e > 0$.

For higher level orderings, where the length of our query is q_l, keeping the ordering of nne and ngg as in the figure, the partial orderings must satisfy the following series of inequalities: $n > g$, $g > e$, $n > e$, $n > 2 \cdot g - e$, $\ldots n > q_l \cdot g - (q_l - 1)e$. Therefore, solving for n, g and e for the general case, and assuming that e takes on a constant value c ($e = c$), we have $g = q_l + c$ and $n = q_l^2 + c$. Observe that this happens to correspond to the original definition of our fitness function, assuming that $e = 1$ and $q_l < 10$, that is, the length of queries is smaller than 10.

3 Algorithmic Refinements

There are several limitations to the original approach, that have lead to the refinement and consequent improvement of our methodology. They can be classified in the following categories: *Fitness function limitations*, *semantic limitations* and *structural limitations*. Let us now describe in detail the changes performed to our algorithms to try to improve on this deficiencies.

3.1 Fitness Function Improvements

The definition of the fitness function given in the previous section has the limitation, that its base b is global to the catalog and therefore, each node is weighted equally independently of its location within the product category tree. This contradicts the usual way product catalogs are structured, because normally, elements found at lower levels in the tree denote more specific products than those found at higher levels. This specificity factor is not taken into account unless we redefine our fitness function to incorporate a base $b_l > 0$ that assigns a greater value to nodes found at lower levels:

Definition ((Level) Fitness Function) Let q_i be the current subquery to be evaluated, and C_i its associated input context. Each node $n \in C_i$ is augmented to have a value v_n resulting from the evaluation of the previous subqueries $q_0 \ldots q_{i-1}$. Then, the fitness function assigns a value v_m to each new node in $m \in C_{i+1}$ generated by a node $n \in C_i$, such that: if $m \in C_{i+1}^n$, then $v_m = b_l^2 + v_n$; if $m \in C_{i+1}^g$, then $v_m = b_l + v_n$; and if $m \in C_{i+1}^e$, then $v_m = 1 + v_n$. $b_l > 0$ is a level-dependent positive integer that represents the value of the fitness function base at each level in the product catalog. □

This small change has the particular effect of increasing the number of levels the query partial order has since, for example in the ordering of Figure 1, nge and gne would be assigned to different levels. Following the same reasoning, however, we could argue that some catalogs contain nodes at the same level that should be weighted differently. Therefore, a more involved representation of the fitness function, but also more flexible, would allow b, the base of the fitness function, to be dependent on the specific node it is applied

upon:

Definition ((Node) Fitness Function) b, the base of the fitness function, is now dependent on the node. Therefore, $b = f(m)$, where $f(m) > 0$ is a node-specific function that returns a positive integer that represents the value of the fitness function base for the node m in the catalog. □

Again, this definition has the effect of thinning out the query partial order, providing us with extra information that allows to distinguish between the different permutations at a particular level in the ordering. The implementation of such a function is not so straightforward, since we need a way to determine the weight of each node. The two strategies that come to mind can be classified as follows:
Hand-crafted functions: Where the maintainer of the local catalog decides based on experience what the weight for each node should be. *Learning functions:* Where the function modifies itself based on the kind of queries performed on the local catalog.

Finally, the fitness function could take into consideration the structure of the global catalog. Since the purpose of our algorithm is to avoid any kind of communication between the global and the local catalogs except for the specific query that needs to be evaluated, the only way for a local catalog to incorporate any kind of information from the global catalog is through analysis of the query received to be evaluated. Therefore, the final version of the fitness function we consider in this paper, is a query-specific fitness function that assigns a higher weight to subqueries further down in the query specification.

Definition ((Query) Fitness Function) b, the base of the fitness function, depends on the position of a specific subquery. Therefore, $b = f(q_i)$, where $f(q_i) > 0$ is a query-specific function that returns a positive integer representing the value of the fitness function base for the ith subquery, such that: $f(q_i) < f(q_j) \iff 0 <= i < j <= n$ □

Therefore, this definition of the fitness function has the effect of transforming our query partial order into a *total linear order*, where each pair of query evaluations can be compared to each other.

3.2 Semantic Extensions

There is a class of "problematic nodes" in product catalogs (from this point on called *semantically weak nodes*), that, under certain conditions, lead to non-optimal results and that, although resolved by the application of a correct node-specific fitness function, can be explained best by referring to semantic information.

Semantically weak nodes are nodes that by themselves do not convey enough information to be a distinguishing product, since there are other nodes in the system with the same name. For example, it does not make any sense to talk about `accessories` without knowing whether or not we are talking about `CPU accessories`, or `printer accessories`, etc. By assigning a smaller value for the fitness function base evaluated on such nodes, we can improve on the results obtained at such nodes, without negatively affecting the outcome of other queries, as it has been verified by the experiments detailed in section 4.

3.3 Improved Structural Adaptability

Since the original implementation of our algorithm only takes into consideration element nodes, and not attributes, we need to incorporate a new type of transformation that allows us to also include attributes in our catalog search: the subquery *upgrade* transformation. An attribute node needs to be upgraded, that is, replaced by its corresponding parent element, if there is a match between either the name or the value of such attribute and the specification of a particular subquery.

The application of this operation allows us to correctly evaluate the XPath query $Q_{global} = /a/b/c$ by effectively translating it to $/a[d = \text{'}b\text{'}]/c$ or $a[b = \text{'*'}]/c$ as needed.

4 Experimental Evaluation

In order to prove the feasibility of our approach, we have performed experiments on top of reduced versions of real XML catalogs provided by several product resellers on the Internet.

All experiments have been performed on a Pentium class computer running Linux and the OpenLDAP [2] server. This query processing system, described in [ML01], is able to process XPath queries on top of arbitrary XML documents stored in an LDAP [WHK97] database, but our results are independent of the actual storage mechanism used and are, therefore, only a function of the adaptive algorithms explained throughout this paper.

In order to test our algorithms, we have visited the homepages of Reichelt (*Rei.*), Alternate (*Alt.*) and K & M Elektronik (*K & M*) [3], three real-world companies that sell electronics and computer products online, and constructed XML catalogs from the data found in their site. The fourth catalog used in the experiments is an artificially crafted catalog (*Art.*), that heavily uses attributes to encode the type of information that is found in the other three catalogs under different elements and product categories. Finally, we have combined their DTDs into a virtual global catalog that contains no data, by inspecting the products offered by each company, and providing a product hierarchy that encompasses all products. Table 1 contains some structural data for each catalog. The column "Nodes" represents the total number of nodes in the catalog, "Depth" the maximum length from the root to the leaves, and "Outdeg", the maximum number of children a particular node has.

Catalog	Nodes	Depth	Outdeg
Global	129	4	16
Alt.	299	5	18
Rei.	89	5	9
K & M	136	4	22
Art.	27	3	6

Table 1: Catalog Sizes and Structures

Type of Fitness Function	Semantic	
	No	Yes
Global, $b = 10$	GN	GS
Level-specific, $1 \leq b \leq 10$	LN	LS
Node-specific (learning), $1 \leq b \leq 10$ (obtained by averaging the frequency of queries)	NN	NS
Query-specific, $1 \leq b \leq 10$ (determined by the position of a specific subquery)	QN	QS

Table 2: Types of Fitness Functions

[2] http://www.openldap.org
[3] http://www.reichelt.de, http://www.alternate.de, http://www.kmelektronik.de

4.1 Adaptive Algorithm Testing

The purpose of this set of experiments is to determine the level of correctness of our algorithms, given that they use the types of fitness functions and semantic extensions detailed in the previous sections.

	No semantic					**Semantic**				
	Alt.	Rei.	K & M	Art.	**Overall**	Alt.	Reich.	K & M	Art.	**Overall**
Global:	2.8%	0.3%	4.8%	1.8%	**2.4%**	0.6%	0.3%	2.6%	0.0%	**0.9%**
Level:	2.7%	0.3%	2.6%	1.8%	**1.8%**	0.5%	0.3%	2.6%	0.0%	**0.9%**
Query:	2.4%	0.1%	3.1%	1.8%	**1.8%**	0.1%	0.3%	0.5%	0.9%	**0.5%**
Node:	2.3%	0.1%	3.4%	1.0%	**1.7%**	0.4%	0.3%	1.3%	0.0%	**0.4%**

Table 3: Error Rates

Table 2 shows the types of fitness-functions which we applied in our experiments. We evaluated all of these fitness-functions with the semantic extension (*semantic=yes*) described in section 3.2; and without it (*semantic=no*). The functions are able to perform the transformations described in [LM02], plus the *upgrade* operation described in section 3.3.

Table 3 shows the error rate for each type of function detailed by catalog and the overall error rates found by performing 783 queries on top of the four local test catalogs. In it we can see that the error rates vary between 3.5% and 0.4%, with accuracy going up to 100% for all catalogs if we consider the second highest rated nodes also part of the solution.

In a second set of experiments, we submitted to three of the four local catalogs every possible query that could be generated by the remaining catalog. As it can be seen in table

	GS Function				**QS Function**			
	Alt.	Rei.	K & M	Art.	Alt.	Rei.	K & M	Art.
Alt:	–	1.1%	2.3%	0.9%	–	0.3%	0.7%	0.9%
Reich:	4.8%	–	2.4%	0.8%	1.4%	–	0.4%	0.8%
K&M:	1.6%	0.0%	–	0.0%	0.2%	0.0%	–	0.7%
Art:	0.0%	0.0%	6.0%	–	0.0%	0.0%	0.0%	–

Table 4: Cross-Evaluation Catalog Error Rate

4, the error rate for two of the tested functions (GS and QS), indicates that our algorithms present exactly the same behavior and incur in approximately similar error rates.

4.2 Error classification

The types of queries performed on the system that do not lead to an optimal result are of two types: (1) Queries performed on *semantically weak nodes* where the catalog lacks specific products that exactly match those of the weak node; and (2) queries whose correct result is debatable, that is, not even a theoretically perfect rewriting algorithm would be able to automatically determine whether or not the solution proposed by our algorithm is correct without the catalog maintainer making a somewhat informed decision.

The first type of errors can be corrected by incorporating some type of semantic information to direct the search, whereas the second type of errors must be corrected by the appropriate tweaking of the fitness function for those queries/nodes that do not produce

the desired result. The flexibility of the fitness function allows for such modifications without the need to change the mechanism of our algorithms.

4.3 Result Analysis

From the two result tables, we can conclude that our algorithm is very insensitive to structural differences. Even though the topology of the *Alternate* catalog is much closer to that of the global catalog than the *Reichelt* catalog, the error rate for *Reichelt* is significantly smaller. The main reason is the absence of *semantically weak nodes* in the *Reichelt* catalog, leading us to conclude that the accuracy of our algorithm is dependent on the clean and clear specification of product names and categories.

Furthermore, the results obtained from the second set of experiments performed on our catalogs, indicate that from a purely technical perspective, there is no need to have a global catalog to centralize our queries, since our algorithms are able to deal with the most varied structural differences without a degradation in performance, as measured by the error rate.

5 Related Work

The integration of catalogs is discussed, from a very broad perspective in [SH01], where the authors point out some of the challenges integration software is usually faced with. The authors indicate that integration software should be based on XML and be able to process XPath queries now, and possibly XQuery in the future.

Similar goals for the easy evaluation of queries on top of XML documents are described in [SKW01], where the authors describe the idea of nearest concept queries to allow for query processing on documents whose mark-up structure is not known. The difference lies on the specifics of their methodology, since they involve the use of a *meet* operator and the evaluation of regular path expressions, whereas our algorithms exploit query transformations to achieve their goal.

Interestingly enough, the need for additional query formulation techniques have lead other researches towards also using regular path expressions [AQM+97, FS98], but in our opinion, a graphical interface for user navigation within the tree structure would just suffice for these purposes, given the adaptive evaluation techniques described in our work, and that typical catalog users cannot be expected to write regular path expressions on their own.

6 Conclusion

In this paper we have extended, improved and evaluated a set of adaptive XPath evaluation techniques by including the processing of semantically weak nodes, as well as the definition of a new type of transformation that allows us to transform attributes into elements.

We have also provided a formal characterization of our fitness function definitions, as well as experimental results that show the applicability of our adaptive algorithm on real-world data, while at the same time, categorizing the types of errors our system could produce if certain unfavorable conditions are met, so that catalog maintainers can avoid them. Furthermore, we have reached the conclusion that, from a strictly technical point of view, the role of the global catalog could be assumed by any local catalog.

To the best of our knowledge, there is no other XPath evaluation system used for XML-based catalog integration that is able to adapt to the specific structure of the catalogs in use without requiring some form of a rewriting algorithm to transform a query performed on a structurally different catalog, while still providing the level of accuracy our system does.

References

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digial Libraries*, 1(1):68–88, 1997.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. `http://www.w3c.org/TR/xpath`, November 1999.

[FLM98] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database Techniques for the World-Wide-Web: A Survey. *Sigmod Record*, 27(3), September 1998.

[FS98] Mary F. Fernández and Dan Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In *Proc. of the 14th Intl. Conf. on Data Engineering*, Orlando, FL., USA, Feb. 1998.

[Jhi00] A. Jhingran. Moving up the food chain: Supporting E-Commerce Applications on Databases. *Sigmod Record*, 29(4), December 2000.

[Kel97] A.M. Keller. *Readings in Electronic Commerce*, chapter Smart Catalogs and Virtual Catalogs. Addison Wesley, 1997.

[LM02] Georg Lausen and Pedro José Marrón. Adaptive Evaluation Techniques for Querying XML-based E-Catalogs. In *Proc. of the 12th Intl. Workshop on Research Issues on Data Engineering*, February 2002.

[ML01] Pedro José Marrón and Georg Lausen. On Processing XML in LDAP. In *Proc. of the 27th Intl. Conf. on Very Large Data Bases (VLDB)*, Rome, Italy, September 2001.

[SH01] Michael Stonebraker and Joseph M. Hellerstein. Content Integration for E-Commerce. In Walid G. Aref, editor, *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, Santa Barbara, California, May 2001.

[SKW01] Albrecht Schmidt, Martin Kersten, and Menzo Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proc of the 17th Intl. Conf. on Data Engineering*, Heidelberg, Germany, April 2001. IEEE Computer Society.

[W3C] W3C – The World Wide Web Consortium. `http://www.w3c.org/`.

[WHK97] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). RFC 2251, Dec, 1997.