Query Optimization for Semistructured Data*

Jason McHugh, Jennifer Widom
Stanford University
{mchughj,widom}@db.stanford.edu, http://www-db.stanford.edu

Abstract

XML is an emerging standard for data representation and exchange on the World-Wide Web. Due to the nature of information on the Web and the inherent flexibility of XML, we expect that much of the data encoded in XML will be *semistructured*: the data may be irregular or incomplete, and its structure may change rapidly or unpredictably. This paper describes the query processor of *Lore*, a DBMS for XML-based data supporting an expressive query language. We focus primarily on Lore's cost-based query optimizer. While all of the usual problems associated with cost-based query optimization apply to XML-based query languages, a number of additional problems arise, such as new kinds of indexing, more complicated notions of database statistics, and vastly different query execution strategies for different databases. We define appropriate logical and physical query plans, database statistics, and a cost model, and we describe plan enumeration including heuristics for reducing the large search space. Our optimizer is fully implemented in Lore and preliminary performance results are reported.

1 Introduction

The World-Wide Web community is rapidly embracing *XML* as a new standard for data representation and exchange on the Web [BPSM98]. At its most basic level, XML is a document markup language permitting tagged text (*elements*), element nesting, and element references. However, XML also can be viewed as a data modeling language, and a significant potential user population views XML in this way [Mar98]. Fortuitously, work from the database community in the area of *semistructured data* [Abi97, Bun97]—work that significantly predates XML—uses graph-based data models that correspond closely to XML. Thus, research results in the area of semistructured data are now broadly applicable to XML.

Semistructured data has been defined as data that may be irregular or incomplete, and whose structure may change rapidly or unpredictably. Although data encoded in XML may conform to a *Document Type Definition*, or *DTD* [BPSM98], DTD's are not required by XML. Due to the nature of information on the Web and the inherent flexibility of XML—with or without DTD's—we expect that much of the data encoded in XML will exhibit the classic characteristics of semistructured data as outlined above.

The *Lore* system at Stanford is a complete DBMS designed specifically for semistructured data [MAG+97]. Lore's original data model, the *Object Exchange Model* (*OEM*), is a graph-based data model with a close correspondence to XML. The query language of Lore, called *Lorel* (for *Lore Language*), is an expressive OQL-based language for declarative navigation and updates of OEM databases. We are in the process of migrating Lore to a fully XML-based data model and query language, but the results presented in this paper are changing very little as the migration occurs.

This paper describes Lore's query processor, with a focus on its cost-based query optimizer. While our general approach to query optimization is typical—we transform a query into a *logical*

*This work was supported by Rome Laboratories under Air Force Contract F30602-96-1-0312 and by the National Science Foundation under grant IIS-9811947.

query plan, then explore the (exponential) space of possible *physical plans* looking for the one with least estimated cost—a number of factors associated with XML data complicate the problem. Path traversals (i.e., navigating subelement and reference links) play a central role in query processing and we have introduced several new types of indexes for efficient traversals through data graphs. The variety of indexes and traversal techniques increases our search space beyond that of a conventional optimizer, requiring us to develop aggressive pruning heuristics appropriate to our query plan enumeration strategy. Other challenges have been to define an appropriate set of statistics for graph-based data and to devise methods for computing and storing statistics without the benefit of a fixed schema. Statistics describing the "shape" of a data graph are crucial for determining which methods of graph traversal are optimal for a given query and database.

Once we have added appropriate indexes and statistics to our graph-based data model, optimizing the navigational *path expressions* that form the basis of our query language does resemble the optimization problem for path expressions in object-oriented database systems, and even to some extent the join optimization problem in relational systems. As will be seen, many of our basic techniques are adapted from prior work in those areas. However, we decided to build a new overall optimization framework for a number of reasons:

- Previous work has considered the optimization of single path expressions (e.g., [GGT96, SMY90]). Our query language permits several, possibly interrelated, path expressions in a single query. We have developed an optimization framework that operates on a set of path expressions, as well as path expressions in the context of a complete query with other query constructs.

- The statistics maintained by relational DBMS's (for joins) and object-oriented DBMS's (for path expression evaluation) are generally based on single joining pairs or object references, while for accuracy in our environment it is essential to maintain more detailed statistics about complete paths.

- The capabilities of deployed OODBMS optimizers are fairly limited, and we know of no available prototype optimizer flexible enough to meet our needs. Building our own framework has allowed us to experiment with and identify good search strategies and pruning heuristics for our large plan space. It also has allowed us to integrate the optimizer easily and completely into the existing Lore system.

2 Related Work

Lore. The first version of the Lorel query language and initial ideas for the Lore system were presented in [QRS+95]. Details of the syntax and semantics of the current version of Lorel can be found in [AQM+97]. The overall architecture of the Lore system, including the simple query processing strategy we used prior to developing our cost-based query optimizer, can be found in [MAG+97].

Other semistructured databases. The *UnQL* query language [BDHS96, FS98] is based on a graph-structured data model similar to OEM. For query optimization, a translation from UnQL to *UnCAL* is defined [BDHS96], which provides a formal basis for deriving optimization rewrite

rules such as pushing selections down. However, UnQL does not have a cost-based optimizer as far as we know. The *Strudel* Web-site management system is based on semistructured data [FFLS97, FFK+99], and query optimization is considered in [FLS98]. In Strudel, semistructured data graphs are introduced for modeling and querying, while the data itself may reside elsewhere in arbitrary format. A key feature of Strudel's approach to query optimization is the use of declarative *storage descriptors*, which describe the underlying data stores. The optimizer enumerates query execution plans, with a cost model that derives the costs of operators from their descriptors. In contrast, Lore data is stored under our control, and the user may dynamically create indexes to provide efficient access methods depending upon the expected queries. Finally, [FLS98] includes detailed experimental results of how large their search space is, but no other performance data is given. In contrast, our experiments focus on the performance of the query plan selected by the optimizer versus other possible query plans.

Some much earlier systems, such as *Multos* [BRG88] and *Model 204* [O'N87], considered problems associated with semistructured data but in very different settings. Multos operated on *complex data objects* which allowed, among other things, sets and pointers to objects of any type. Basic knowledge of the schema was crucial, however, and queries were placed into categories with a fixed set of execution strategies for each category. Lore follows a more traditional and flexible model of query processing. Model 204 was based on self-describing record structures somewhat resembling OEM, but the work concentrated primarily on clever bit-mapped indexing structures to achieve high performance for its relatively simple queries.

Relational databases. As mentioned earlier, at a coarse level the problem of optimizing a Lorel path expression is similar to the join ordering problem in relational databases. However, join ordering algorithms usually rely on statistics about each joining pair, while for typical queries in our environment it is crucial to maintain more comprehensive statistics about entire paths. The computation and storage of our statistics is further complicated by the lack of a schema. In addition, when quantification is present in our queries, the SQL translation results in complex subqueries that many relational optimization frameworks are ill-equipped to handle.

Object-oriented databases. Many of the points discussed in the previous paragraph apply to object-oriented databases as well. There has been some work on optimizing path expressions in an OODBMS context [GGT96]. They propose a set of algorithms to search for objects satisfying path expressions containing predicates, and analyze their relative performance. Our work differs in that we consider many interrelated path expressions within the context of an arbitrary query with other language constructs. We also provide additional access methods for path expressions, and our optimization techniques are implemented within a complete DBMS. Similar comparisons can be drawn between our work and other recent OODB optimization work, e.g., [GGMR97, KMP93, OMS95, SO95]. Some recent papers have specified cost models for object-oriented DBMSs, e.g., [BF97, GGT95]. Object-oriented databases typically support object clustering and physical extents, rendering many of these formulas inapplicable in our setting.

Generalized path expressions. Other recent work, including [FLS98] discussed above, has considered the problem of optimizing the evaluation of *generalized path expressions*, which describe traversals through data and may contain regular expression operators. In [CCM96] an algebraic framework for the optimization of generalized path expressions in an OODBMS is proposed, including an approach that avoids exponential blow-up in the query optimizer while still offering

flexibility in the ordering of operations. In [FS98] two optimization techniques for generalized path expressions are presented, *query pruning* and *query rewriting using state extents*. Lore's techniques for handling generalized path expressions are covered in [MW99] and are not the focus of this paper.

XML query languages. The *XML-QL* data model and query language [DFF+98] is similar in expressibility to ours, with some extensions specific to the current specification of XML. *XQL* [RLS98] is a simpler query language based on single path expressions and is strictly less powerful than XML-QL, Lorel [AQM+97], StruQL [FFLS97], or UnQL [BDHS96]. To the best of our knowledge no full cost-based query optimizer has been developed for XML-QL or XQL, and the optimization principles presented in this paper should be directly applicable when that task is tackled.

3 Preliminaries

3.1 Data Model

Lore's original data model, *OEM* (for *Object Exchange Model*) [PGMW95], was designed specifically for semistructured data. Data in OEM is schema-less and self-describing, and can be thought of as a labeled directed graph. For example, the very small OEM database in Figure 1 contains (fictitious) information about the Stanford Database Group. The vertices in the graph are *objects*; each object has a unique *object identifier* (oid), such as &5. *Atomic objects* have no outgoing edges and contain a value from one of Lore's basic atomic types such as `integer`, `real`, `string`, `gif`, `java`, `audio`, etc. All other objects may have outgoing edges and are called *complex objects*. Object &3 is complex and its *subobjects* are &8, &9, &10, and &11. Object &7 is atomic and has value "Clark". *Names* are special labels that serve as aliases for single objects and as entry points into the database. In Figure 1, *DBGroup* is a name that denotes object &1.

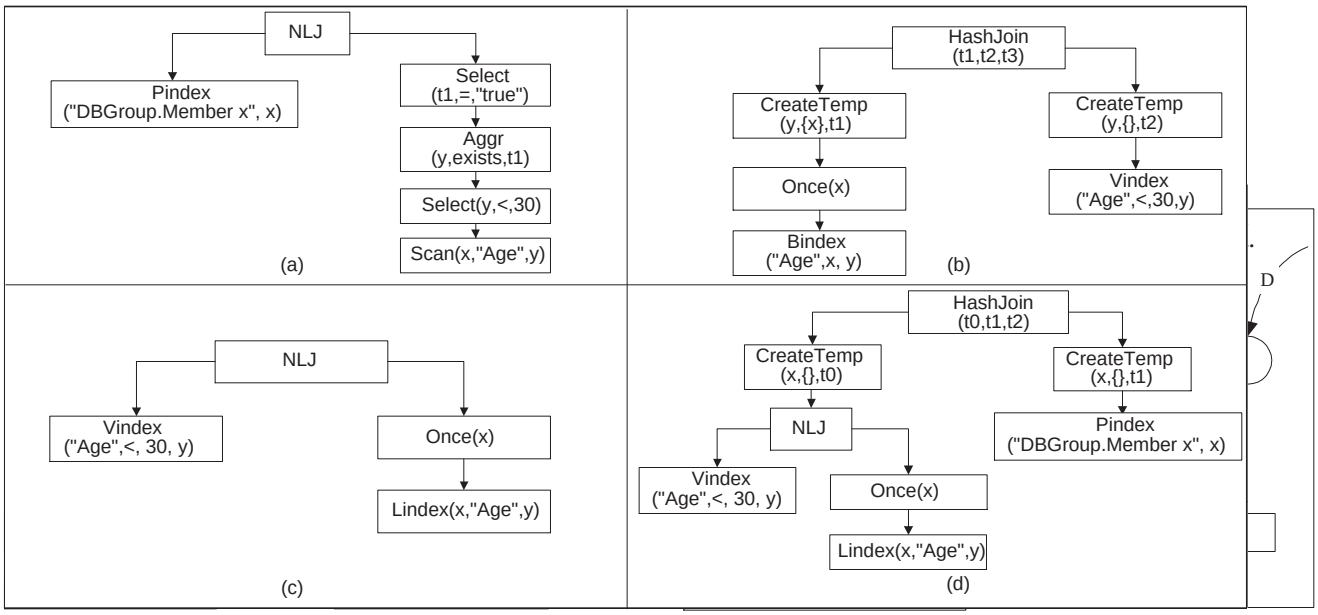
The correspondence between OEM and XML is evident: OEM's objects correspond to elements in XML, and OEM's subobject relationship mirrors element nesting in XML. The fundamental differences are that subelements in XML are inherently ordered since they are specified textually, and XML elements may optionally include a list of attribute-value pairs. Note that graph structure (multiple incoming edges) must be specified in XML with explicit references, i.e., via *ID* and *IDREF(S)* attributes [BPSM98]. The following XML fragment corresponds to the rightmost `Member` in Figure 1, where `Project` is an attribute of type IDREFS.

```
<Member Project="&5 &6">
   <Name>Jones</Name>
   <Age>46</Age>
   <Office> <Building>Gates</Building> <Room>252</Room> </Office>
</Member>
```

As mentioned earlier, we are migrating Lore to use a native XML data model instead of OEM, but most results in this paper carry over to the new version of Lore with few modifications.

We now introduce two definitions that are useful in the remainder of the paper.

Definition 3.1 (**Simple Path Expression**) A *simple path expression* specifies a single-step navigation in the database. A simple path expression for a variable or name x and label l has the form


NLJ
Pindex
("DBGroup.Member x", x)
Select
(t1,=,"true")
Aggr
(y,exists,t1)
Select(y,<,30)
Scan(x,"Age",y)
(a)
HashJoin
(t1,t2,t3)
CreateTemp
(y,{x},t1)
CreateTemp
(y,{},t2)
Once(x)
Vindex
("Age",<,30,y)
Bindex
("Age",x, y)
(b)
NLJ
Vindex
("Age",<, 30, y)
Once(x)
Lindex(x,"Age",y)
(c)
HashJoin
(t0,t1,t2)
CreateTemp
(x,{},t0)
CreateTemp
(x,{},t1)
NLJ
Pindex
("DBGroup.Member x", x)
Vindex
("Age",<, 30, y)
Once(x)
Lindex(x,"Age",y)
(d)
D