Pathfinder:
A Relational Query Optimizer Explores XQuery Terrain

Jan Rittinger Jens Teubner Torsten Grust

Technische Universität München, Institut für Informatik
{jan.rittinger,jens.teubner,torsten.grust}@in.tum.de

1 Purely Relational XQuery

Relational encodings of the *static* aspects of the XQuery data model, *i.e.*, tabular representations for XML documents and ordered sequences of items, are widely used today. Since 2002, the *Pathfinder* and *MonetDB/XQuery* companion projects [BGvK+06] pursue the primary goal to also embrace the complete *dynamic semantics* of XQuery (expression evaluation and runtime aspects) with the help of relational database systems.

In earlier work [GT04], we have shown that relational algebra makes for a suitable target language in an XQuery compiler. This purely relational approach to XQuery inherits the scalability advantages of the underlying relational database back-end and makes proven optimization techniques immediately applicable to the construction of XQuery processors. *MonetDB/XQuery*, an open-source system that implements this approach, is found among the fastest and most scalable XQuery processors available today [BGvK+06].

This is a demonstration of the relational optimizer of *Pathfinder*[1], the query compiler behind MonetDB/XQuery. To account for the significant size and unusual shape of the relational query plans (see Figure 1) derived from input XQuery expressions, Pathfinder implements various optimization techniques in a *peephole-style* fashion and provides support for *graph-shaped plans* from the ground up.

2 Relational Query Optimization in an XQuery Compiler

Pathfinder's XQuery compiler turns incoming XQuery expressions into relational query plans according to the *loop-lifting* compilation strategy we devised in [GT04]. In a nutshell, loop-lifting trades iteration (esp. the XQuery FLWOR construct) for efficient bulk-oriented processing. The compiler emits expressions of a relational algebra whose operators have been chosen to match the actual capabilities of modern SQL query engines. A few representative operators are shown in Table 1 (note that non-standard operators like the XPath step join ◿ are synonyms for relational "micro-plans" with an optimized implementation in Pathfinder's back-end database system MonetDB).

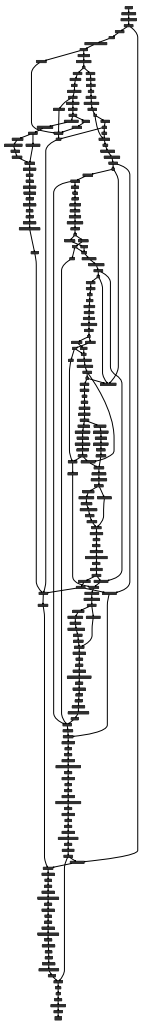
[1] MonetDB/XQuery and Pathfinder are available via `http://www.pathfinder-xquery.org/`.

To correctly reflect the complex XQuery semantics—different notions of *order*, side-effecting *node construction* and *node identity*, *existential predicate semantics*, and implicit *casting* and *atomization*—the resulting plans are of significant size (50–500 operators). Figure 1 gives an impression of the relational plan for XMark query *Q8* [SWK+02] along with the total number of plan operators for each of the XMark queries.

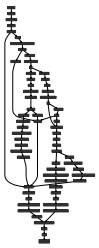
Loop-lifted evaluation plans typically exhibit plenty of opportunities to *share* common subexpressions. Pathfinder represents query plans as *directed acyclic graphs (DAGs)* to account for this sharing. XQuery is a compositional expression-oriented language in which subexpressions are stacked upon each other to form complex queries. Note how the stretched shape of the plans in Figure 1—which is somewhat different from the well-known SQL-induced π–σ–$\bowtie$ query pattern—reflects this compositionality.

2.1 Rewriting Large Plan DAGs

To obtain a bird's eye view of these large plans and maintain focus during optimization, Pathfinder identifies *basic blocks* (straight-line operator sequences in the DAG with no sideways entries), much like compilers for programming languages. A further technique that saves the optimizer from getting lost in large plan DAGs is *peephole-style* inspection that considers one operator at a time. To compensate for this restricted peephole view, a *property inference* phase precedes the actual rewriting. The inference carries additional information about the relevant vicinity of each plan node into the operator itself. We highlight some of these inferred plan characteristics in the following.



(a) Original plan.



(b) Optimized plan.

Q	(a)	(b)
1	138	37
2	91	34
3	385	135
4	325	67
5	132	40
6	53	23
7	79	46
8	227	66
9	371	101
10	500	243
11	234	91
12	316	110
13	87	43
14	86	54
15	100	31
16	149	51
17	97	42
18	101	48
19	142	76
20	392	147

Figure 1: Plans for XMark query *Q8*, (a) before and (b) after relational optimization. Table: Operator counts for the 20 XMark queries.

π	column projection, renaming	ϱ	row numbering
σ	row selection	$\dot\cup$	disjoint union (`UNION ALL`)
$\bowtie$	equi-join	$\circledcirc$	arithmetic/comparison operator $\circ$
$\times$	Cartesian product	$\scriptstyle\diagup\!\!\diagup$	XPath step join

Table 1: Subset of the relational algebra emitted by the loop-lifting compiler. Operator ϱ is the equivalent of SQL:1999's `ROW_NUMBER` operator—see [GT04] for details.

2.2 Column Pruning

Pathfinder's fully compositional compilation may lead to relational plans that generate table columns whose contents may not be relevant to the semantics of a given query. Sequence order (maintained in column pos [GT04]), for example, is not relevant to compute the result of the XQuery general comparison operators (=, <, ...). To avoid the runtime overhead incurred by the generation and maintenance of such columns, Pathfinder annotates each relational plan operator with the set of columns that is *strictly required* to process its upstream plan. These annotations are then used to drive a variant of *projection pushdown* [Gru05, JK84].

Loop-lifted XQuery evaluation plans are very susceptible to this optimization. Columns carrying the type annotations derived by XQuery's `validate{}` construct, for example, will only be retained in the optimized plans if the surrounding XQuery expressions indeed inspect these types (*e.g.*, via `typeswitch` or `instance of`). Similarly, we are currently extending Pathfinder's internal data model with a score column intended to support XQuery Full-Text retrieval [AYBB+06]. Column pruning will ensure that this will not negatively affect the performance of queries that do not use XQuery Full-Text features.

2.3 Functional and Multi-Valued Dependencies

Due to loop-lifting, optimization opportunities like the occurrence of invariant subexpressions inside (deeply nested) XQuery FLWOR blocks or the presence of value-based joins, surface as *functional* and *multi-valued dependencies* in the relational plans. During the property inference phase, Pathfinder derives the validity of degenerate functional dependencies of the form $\varnothing \rightarrow c$ for all columns c produced by sub-plans of the query DAG. If $\varnothing \rightarrow c$ holds for a relation, all its rows carry the same value in column c—an indicator for Pathfinder's optimizer to initiate constant propagation and folding in the plan vicinity. Likewise, the presence of a degenerate multi-valued dependency $\varnothing \twoheadrightarrow c_1, \ldots, c_n$ in a relation signals that its columns $c_1, \ldots, c_n$ are independent of all its remaining columns. In a loop-lifted query plan, this multi-valued dependency is the relational expression of the fact that a loop-invariant computation occurs inside an XQuery `for`-iteration. In such cases, Pathfinder performs a variant of loop hoisting to improve the original plan.

Note that relational dependency analysis is indifferent to XQuery's syntactic diversity—in XQuery syntax, value-based joins are not as prominent as in SQL and come in various flavors—and will detect the value-based join in `let $d := fn:doc(···) for $a in $d//a return $d//b[@c = $a/@d]`, for example.

2.4 Data Flow Analysis Based on Active Domains

Pathfinder's join recognition logic is further supported by *data flow analysis* on relational plan DAGs. For all intermediate result tables t, the system infers an approximation α_c of the *active domain* for each column c in t [Klu80]. The data flow may then be inferred from the inclusion (or disjointness) of these active domains: for two columns c_1 and c_2 of arbitrary intermediate result tables, the inclusion $\alpha_{c_1} \subseteq \alpha_{c_2}$ indicates data flow between

the respective plan operators.

Among many other uses, data flow analysis helps to maintain the correspondence between the pair of branches that results from the compilation of an XQuery `if-then-else` clause: the active domain relationships remain intact even if extensive rewrites move the `then` and `else` DAG branches far apart. Further, active domain information is an essential building block of a procedure that provides cardinality estimates for arbitrary XQuery subexpressions (not just the overall query result).

3 Demonstration Setup

We demonstrate an instance of MonetDB/XQuery against which users may run arbitrary queries in an interactive fashion. The system will be preloaded with various XMark XML instances (of 100 KB–1 GB serialized size). Most importantly, users will be able to look under the hood of the Pathfinder compiler and experience the effect of the aforementioned relational optimization techniques. The demonstration system renders the relational query plans (much like the plans in Figure 1) to enable the inspection of plan characteristics at various stages of Pathfinder's highly-configurable optimizer pipeline. Stages may be separately enabled to judge their impact on plan quality and XQuery evaluation performance.

Acknowledgment. This research is supported by the DFG (Deutsche Forschungsgemeinschaft) under grant GR 2036/2-1. We thank the MonetDB development team at CWI, Amsterdam, for the lasting and fun collaboration.

References

[AYBB+06] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, D. McBeath, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 Full-Text. W3C Working Draft, May 2006.

[BGvK+06] P. Boncz, T. Grust, M. v. Keulen, S. Manegold, J. Rittinger, and J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In *Proc. of the 2006 ACM SIGMOD Conf.*, Chicago, IL, USA, June 2006.

[Gru05] T. Grust. Purely Relational FLWORs. In *Proc. of the 2nd Int'l XIME-P Workshop*, Maryland, MD, USA, June 2005.

[GT04] T. Grust and J. Teubner. Relational Algebra: Mother Tongue—XQuery: Fluent. In *Proc. of the 1st Twente Data Management Workshop (TDM)*, Enschede, The Netherlands, June 2004.

[JK84] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, 16(2), June 1984.

[Klu80] A. Klug. Calculating Constraints on Relational Expressions. *ACM TODS*, 5(3), September 1980.

[SWK+02] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. of the 28th Int'l VLDB Conf.*, Hong Kong, China, August 2002.