Analyzing Plan Diagrams of XQuery Optimizers

Bruhathi H S and Jayant R. Haritsa

Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore 560012, INDIA

Abstract. The automated optimization of declarative user queries is a classical hallmark of database technology. XML, with its support for deep data hierarchies and powerful query operators, including regular expressions and sibling axes, renders the query optimization challenge significantly more complex. In this paper, we analyze the behavior of industrial-strength XQuery optimizers using the notion of "plan diagrams", which had hitherto been applied solely to relational engines. Plan diagrams visually characterize the optimizer's query plan choices on a parametrized query space, and extending them to the XML environment requires redesigned definitions of the parameters and the space. Through a comprehensive set of experiments on a variety of popular benchmarks, we demonstrate that XQuery plan diagrams can be significantly more dense and complex as compared to their relational counterparts. Further, they are more resistant to "anorexic reduction", requiring substantially larger cost-increase thresholds to achieve this objective. These results suggest that important research challenges remain to be addressed in the development of effective XQuery optimizers.

1 Introduction

Over the last decade, the flexibly structured XML language has become the de-facto standard for data representation and information exchange between applications. XML data was initially stored in traditional DBMS formats by shredding into relational tuples (e.g. [10]). However, in recent times most database vendors have augmented their SQL engines to provide *native* support for XML storage and XQuery interfaces, resulting in the so-called "hybrid" processors – examples include IBM DB2 [8], Oracle [14] and Microsoft SQL Server [15].

The automated optimization of declarative SQL queries is a classical hallmark of database technology. XML with its support for deep data hierarchies and powerful query operators, including regular expressions and sibling axes, has far more expressive power than SQL. Therefore, the optimization challenge becomes significantly more complex, motivating us to investigate, in this paper, the behavior of industrial-strength XQuery optimizers. For our analysis, we use the notion of "plan diagrams" developed in [9], which had hitherto been applied solely to relational engines, to drive the evaluation. Plan diagrams visually characterize the optimizer's query plan choices over an input parameter space, whose dimensions may include database, query and system-related features. In a nutshell, plan diagrams pictorially capture the geometries of the optimality regions of the *parametric optimal set of plans* (POSP) [5].

For a given database and system setup, the plan choices made by query optimizers are primarily a function of the *selectivities* of the predicates appearing in the query.

Accordingly, our focus here is on plan diagrams obtained through selectivity variations on parametrized XQuery templates. In this process, we have to tackle a variety of questions, including: (1) At what data granularity should the selectivity be varied – specifically, *document* level and/or *node* level?; (2) What is the mechanism for varying selectivities – specifically, through *structural* predicates and/or *value* predicates?; (3) How are selectivities to be reliably estimated from the metadata available in these engines?; and (4) What constraints need to be imposed on the construction of XQuery templates such that the resulting diagrams are semantically meaningful?

We begin this paper by describing our attempts to address the above issues. Subsequently, using the developed framework, we carry out a detailed analysis of a popular commercial hybrid XQuery/SQL optimizer, which we refer to as **XOpt**, through an extensive set of plan diagrams generated on three benchmark environments – XBench [13], TPoX [7] and TPCH_X, the XML equivalent of the classical TPC-H [16] relational benchmark. Our experimental results suggest that even two-dimensional plan diagrams are often extremely dense, featuring hundreds of different plans, and further, exhibiting intricate geometric patterns. This was especially the case when the XQuery template featured *order by* clauses, *wild cards*, and *navigational axes*.

We also observed that when an XQuery template is rewritten in an equivalent XML/SQL format [4], the plan diagrams produced are markedly different, clearly demonstrating that the choice of interface has a sizeable impact on the optimal plans' cost behavior.

Finally, it had been empirically found in the relational world that even complex plan diagrams could be simplified to retain just a few plans with only a marginal impact on the query processing quality – this property was termed "anorexic reduction" in [3], and has several useful applications, including providing robustness to selectivity estimation errors. In the XML environment, however, we find instances wherein achieving anorexic reduction incurs a very substantial deterioration of query processing quality.

Taken in toto, these results suggest that important challenges remain to be addressed in the development of effective XQuery optimizers.

2 Background on Plan Diagrams

To set the stage, we first overview the notion of plan diagrams, developed in [9]. Consider a parametrized SQL query template that defines a relational selectivity space – for example, QT8 shown in Fig. 1, which is based on Query 8 of the TPC-H benchmark. Here, selectivity variations on the SUPPLIER and LINEITEM relations are specified through the s_acctbal :varies and l_ extendedprice :varies predicates, respectively.[1]

The corresponding plan diagram for QT8, produced on a commercial database engine, is shown in Fig. 2(a). In this picture[2], each colored region represents a specific plan, and a set of 42 different optimal plans (from the optimizer's perspective), P1 through P42, cover the selectivity space. The value associated with each plan in the

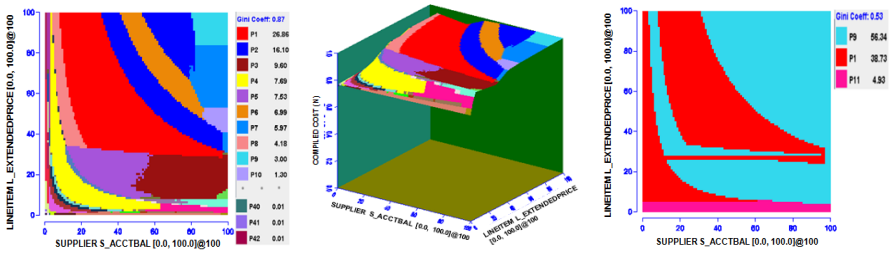
[1] We implement *:varies* using one-sided range predicates of the form *Relation.attribute*$\leq$*const.*

[2] The figures in this paper should ideally be viewed from a color copy, as the grayscale version may not clearly register the features.

```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from (select YEAR(o_orderdate) as o_year, l_extendedprice * (1 - l_discount) as volume, n2.n_name as nation
      from part, supplier, lineitem, orders, customer, nation n1, nation n2, region
      where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey and o_custkey =
          c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey and s_nationkey =
          n2.n_nationkey and r_name = 'AMERICA' and p_type = 'ECONOMY ANODIZED STEEL' and
          s_acctbal :varies and l_extendedprice :varies
      ) as all_nations
group by o_year
order by o_year
```

Fig. 1. Example SQL query template (QT8)

legend indicates the percentage area covered by that plan in the diagram – the biggest, P1, for example, covers about a quarter (26.86%) of the region.



(a) **Plan diagram** (b) **Cost diagram** (c) **Reduced PD (λ = 20%)**

Fig. 2. Plan, cost and reduced plan diagrams (QT8)

Related and complementary to the plan diagram is the "cost diagram", which quantitatively depicts the optimizer's (estimated) query processing costs of the plans featuring in the plan diagram. The cost diagram for the QT8 example is shown in Fig. 2(b).

Plan diagrams are often found to be complex and dense, featuring high plan cardinalities and intricate geometries, as can be observed in Fig. 2(a). However, these dense diagrams can typically be reduced to much simpler pictures retaining only a few plans from the POSP set, while ensuring that the processing quality of *any* individual query is not increased by more than a user-defined threshold λ. For example, the plan diagram in Fig. 2(a) can be reduced to that shown in Fig. 2(c), where only *three* of the original 42 plans are retained, while ensuring that no query suffering a plan replacement has had its cost increased by more than $\lambda = 20\%$. It has been empirically observed in [3] that, for templates based on the TPC-H and TPC-DS relational benchmarks, $\lambda = 20\%$ is typically sufficient to bring the plan cardinality in the final reduced picture down to around *ten plans* or fewer, referred to as "anorexic" (small absolute number) plan diagrams.

3 Generation of XML Plan Diagrams

As mentioned in the Introduction, a variety of issues crop up when we attempt to extend the concept of plan diagrams to the XML world. We discuss our approach to handling these issues in this section.

3.1 Varying XML Selectivity

In XML, information is organized in the form of *nodes* and *documents* containing these nodes. Therefore, selectivities can be computed at the granularity of nodes and/or of documents. For example, consider the scenario where 100 XML nodes are organized in a single document, and the other extreme where there are 100 documents, each containing one of these nodes. In the former case, the document selectivity will always be 0 (no node in the document satisfies the predicate) or 1 (at least one node in the document satisfies the predicate), whereas in the latter, the document selectivity will represent the fractional number of nodes satisfying the predicate. So, variations in the selectivity of XML data can potentially be achieved at the level of documents, or of nodes. In fact, it would even be possible to obtain plan diagrams among which selectivities of different dimensions are varied at different levels. However, in this paper, we only focus on obtaining plan diagrams by varying the selectivity of XML data at the *node level*, since irrespective of how data is distributed – in single or multiple documents – selectivity variations can be brought about in a desired range.

Our next task is to determine the mechanisms for varying the selectivities of XML data. Given a path expression, there are two kinds of selectivities to be estimated – the *structural selectivity* and the *value selectivity*. For example, consider the path `//person/name="Gray"`. Here, estimating the cardinality of the `name` nodes that have a `person` node as their parent corresponds to structural selectivity, and estimating the cardinality of `name` nodes with `"Gray"` as their content corresponds to value-based selectivity. To jointly vary this pair of selectivities in a controlled manner from an external interface is rather complex with today's database engines. However, the operator algorithms that are employed in XOpt are designed to enable the concurrent application of both predicates. We leverage this fact and vary the selectivity of chosen paths in the XML documents, through the application of parametrized value-based predicates to these paths. To ensure that both types of predicates are applied together, we create *value-based indexes* on the required paths. As a case in point, consider the path `/person/name` – here, we vary the path selectivity by applying suitable value-based predicates to `name` nodes that are direct descendants of `person` nodes emanating from the document root, after ensuring that a value-based index is present on the `/person/name` path.

It is to be noted that the above selectivity variations are brought about *externally*, through mechanisms that operate totally outside the database engine. Also, in our current work, we focus solely on varying the selectivities of predicates that are applied to document collections as the first step in the query execution plan. Selectivity variations of join predicates applied between document collections are not considered.

3.2 XQuery Template Construction

An XQuery template is used to specify the paths whose selectivities are to be varied through the application of value-based predicates. We will hereafter use the term **VSX** (Variable Selectivity XML element path)[3] to denote these varying dimensions.

[3] Our usage of the term *element* here denotes both XML elements as well as XML attributes.

An example template is shown in Fig. 3, where there are two VSXs corresponding to customer addresses and customer orders, respectively. This template returns the names and phone numbers of customers located within a (parametrized) zip code and whose orders feature item quantities within a (parametrized) value.

```
1. for $cust in xmlcol(CUSTOMER.CUSTOMER) /customers/customer[address_id :varies]
2.  for $order in xmlcol(ORDER.ORDER)
3.  /order[customer_id=$cust/@id]/order_lines/order_line[quantity_of_item :varies]
4.   return <Res> {$order} {$cust/first_name} {$cust/last_name} {$cust/phone_number} </Res>
```

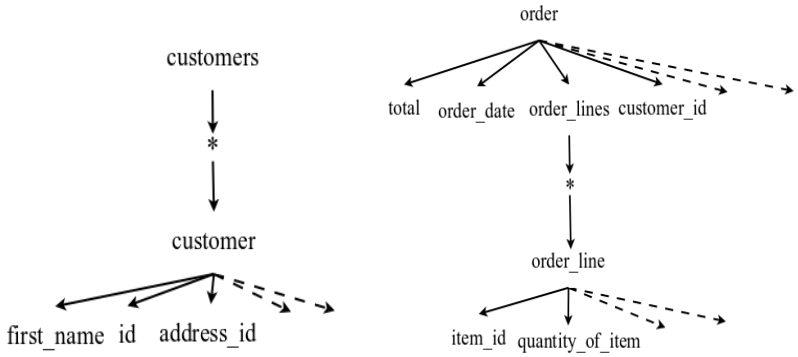
Fig. 3. Example XQuery template

The element path in a VSX predicate typically consists of a logical segment, denoted L, that defines the semantic object whose selectivity is desired to be varied, and a physical segment, denoted P, which is downstream of L and whose value is actually varied. To make this concrete, consider the VSX `/order/order_lines/order_line [quantity_of_item :varies]` in Fig. 3. Here L is the segment `/order/order_lines/order_line`, while P is the downstream segment `order_line/quantity_of_item` – the parametrized variation across `order_lines` is achieved through varying the values of `quantify_of_item`. As a final point, note that it is also acceptable to have templates where the logical segment itself terminates with the variable element, and there is no distinct P segment.

The value constants that would result in the desired selectivities of the VSXs are estimated by essentially carrying out an "inverse-transform" of the statistical summaries (histograms) corresponding to the VSXs. These path-specific value histograms are automatically built by the database engine whenever an index is declared on the associated paths, which as mentioned earlier, is important for our selectivity variation strategy. We employ linear interpolation mechanisms on these summaries to estimate the constants corresponding to the query locations in the selectivity space.

XQuery statements can often be complex, involving powerful constructs from second order logic such as regular expressions and navigational paths. Therefore, it is extremely important that the associated templates be constructed carefully, otherwise we run the risk of producing plan diagrams that are semantically meaningless since the intended selectivity variations were not actually realized in practice. Accordingly, we list below the conditions to be satisfied by a valid XQuery template, arising out of *conceptual* reasons.

1. *Many-to-one* relationships are not permitted to occur in the P segment (if present) of a VSX predicate. This translates to requiring, in the graphical representation of the XML schema [10], that there should be no '*' (wild card) node appearing in the path corresponding to P.
2. A collection C of XML documents adhering to a common schema can participate in at most one VSX. Further, they can also participate in *join-predicates* with other XML document collections, but are not permitted to be subject to additional selection predicates.
3. The VSXs should appear sufficiently frequently in the documents and their value predicates should be over dense domains.



(a) customer Schema Graph (b) order Schema Graph

Fig. 4. XQuery template constraints

To illustrate the constraints on XQuery templates, consider an XML database with Customer and Order document collections adhering to the schema graphs shown in Figs. 4(a) and 4(b), respectively. On this database, consider the XQuery template shown in Fig. 3 – this template is compatible with respect to all of the above conditions. However, if the query template were to be slightly altered to

```
/order[customer_id=$cust/@id and order_lines/order_line/quantity_of_item :varies]
```

in line 3, the template becomes invalid due to violating Requirement 1 above – a '*' node is present in the `order_lines/('*')order_line/quantity_of_item` physical segment.

4 Experimental Results

In this section, we describe the experimental framework on which the XOpt optimizer was evaluated, and discuss the results obtained on a variety of benchmark environments.

4.1 Experimental Setup

Our experiments were carried out on a vanilla hardware platform – specifically, a SUN ULTRA 20 system provisioned with 4GB RAM running Ubuntu Linux 64 bit edition 9.10. The XOpt engine was was run at its default optimization level, after enabling all options that enhanced the scope for optimization.

Databases. We operated with three different XML databases: **XBench**, **TPoX** and **TPCH_X**. TPoX and XBench are native XML benchmarks, while TPCH_X is an XML equivalent of the classical TPC-H relational benchmark. XBench and TPCH_X represent decision-support environments, whereas TPoX models a transaction processing domain – their construction details are given below.

XBench. This benchmark [13] supports four different kinds of databases, of which we chose the **DC/MD** (Data Centric, Multiple Documents) flavor, since it symbolises business data spread over a multitude of documents, and appears the most challenging from

the optimizer's perspective. The DC/MD generator was invoked with the "large" option, resulting in a database size of around 1 GB, with all data conforming to a uniform distribution. For the physical schema, indexes were created for all paths involved in join predicates, and additionally for all paths appearing in VSXs.

TPoX. The data generator of the TPoX benchmark [7] was invoked at the **XXS scale**, resulting in 50000 CUSTACC, 500000 ORDER and 20833 SECURITY documents, collectively taking about 1GB of space. The data in these documents follow a variety of distributions, ranging from uniform to highly skewed. For the physical schema, all the 24 recommended indexes (10 on CUSTACC, 5 on ORDER and 9 on SECURITY) were created. These indexes covered all the paths involved in the join predicates and VSXs featured in our templates.

TPCH_X. Here, the relations of the TPCH [16] benchmark: NATION, REGION, SUPPLIER, CUSTOMER, PART, PARTSUPP, ORDERS, LINEITEM were first converted to their equivalent XML schemas, with ORDER and LINEITEM combined into the Order XML schema, and PART and PARTSUPP merged into Part XML schema. This merging was carried out since individual orders and parts are associated with multiple lineitems and suppliers, respectively, and these nested relationships are directly expressible in XML through its organic support for hierarchies.

Then, the TPC-H relational data at scale 1 was translated to these XML schemas using the Toxgene [1] tool, resulting in a database of around 1 GB size. For the physical schema, indexes were created on all paths that featured in join predicates and in VSXs.

XQuery Templates. For simplicity and computational tractability, we have restricted our attention to *two-dimensional* XQuery templates in this study. These templates were created from representative queries appearing in the above-mentioned benchmarks (X-Bench has a suite of 15 XQueries, TPoX has 11 XQueries and 11 SQL/XML queries, while TPCH_X has 22 XQueries). The templates were verified to be compatible with the constraints specified in Sect. 3, and the VSX value predicates are on floating-point element values (explicit indexes are created on the VSXs, resulting in value based comparisons).

The plan diagrams are produced at a resolution of 300 points in each dimension, unless specified otherwise – this means that close to a hundred thousand queries are optimized in each 2D diagram. Since optimizing an individual query takes between 100 to 200 milliseconds, generating the complete plan diagram typically requires a few hours (3-5 hours). Finally, for plan diagram reduction, which falls into the NP-hard complexity class, we employed the the Cost-Greedy heuristic algorithm described in [3], with the default cost-increase threshold λ set to 20%.

In the remainder of this section, we present results for XQuery plan diagrams produced on the XBench, TPoX and TPCH_X environments.

4.2 Plan Diagrams with XBench

We present here the results for two XQuery templates that cover the spectrum of query complexity: the first, referred to as QTXB1, features the basic constructs, whereas the second, referred to as QTXB2, includes a rich variety of advanced operators.

```
for $order in xmlcol(ORDER.ORDER)/order,
  $cust in xmlcol (CUSTOMER.CUSTOMER)/customers/customer
    where $order/customer_id = $cust/@id and $order/total :varies and $cust/address_id :varies
    order by $order/order_date
    return <Output> {$order/@id} {$order/order_status} {$cust/address_id}
        {$cust/first_name} {$cust/last_name} {$cust/phone_number} </Output>
```

Fig. 5. XQuery template for XBench (QTXB1)

Basic Template. The basic template, QTXB1, is based on Query 19 of the benchmark and is shown in Fig. 5. Its objective is to retrieve all purchases within a (parametrized) total value for which the associated customers are located within a (parametrized) address value, the result being sorted by the purchase date.

The plan diagram for QTXB1 is shown in Fig. 6(a), and we observe that, even for this basic template, as many as 42 plans are present with intricate spatial layouts. Further, the area distribution of the plans is highly skewed, with the largest plan occupying a little over 20% of the space and the smallest taking less than 0.001%, the overall Gini (skew) co-efficient being close to 0.9.

Most of the differences between the plans are due to operator *parameter switches*, rather than the plan tree structures themselves. For example, the difference between plan pairs is often solely attributable to the presence or absence of the TMPCMPRS switch, associated with the TEMP and SORT operators. When such switch differences are ignored, the number of plans comes down sharply to just 10! It is interesting to note that the switch by itself contributes very little to the overall plan cost.

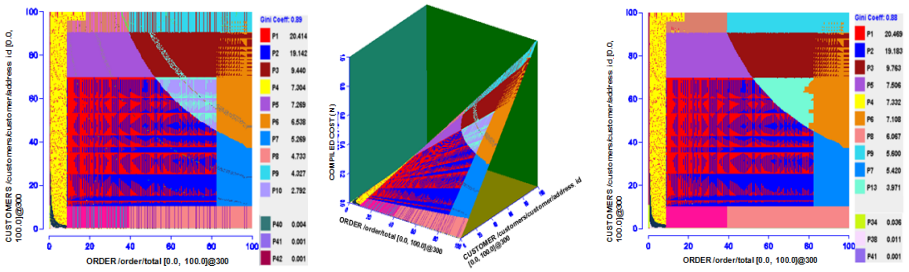
In Fig. 6(a), the plans P1 (red) and P2 (dark blue) blend together in a wave-like pattern. The operator trees for these plans are shown in Figs. 7(a) and 7(b), respectively. We see here that the plans primarily differ on their join orders with respect to the document collections – P1 computes Order $\bowtie$ Customer whereas P2 evaluates Customer $\bowtie$ Order – that is, they differ on which document collection is outer, and which is inner, in the join.

Near and parallel to the Y-axis, observe the yellow vertical strip corresponding to plan P4, sprinkled with light orange spots of plan P16. The P4 and P16 plans differ only in their positioning of an NLJOIN-XSCAN operator pair, where the expressions `$ord/order_status`, `$ord/@id` and `$ord/order_date` of the return clause are evaluated. In XOpt, the XSCAN operator parses the input documents with a SAX parser and evaluates complex XPath expressions in a single pass over these documents.

The associated cost diagram is shown in Fig. 6(b), and we observe a steep and affine relationship for the expected execution cost with regard to the selectivity values.

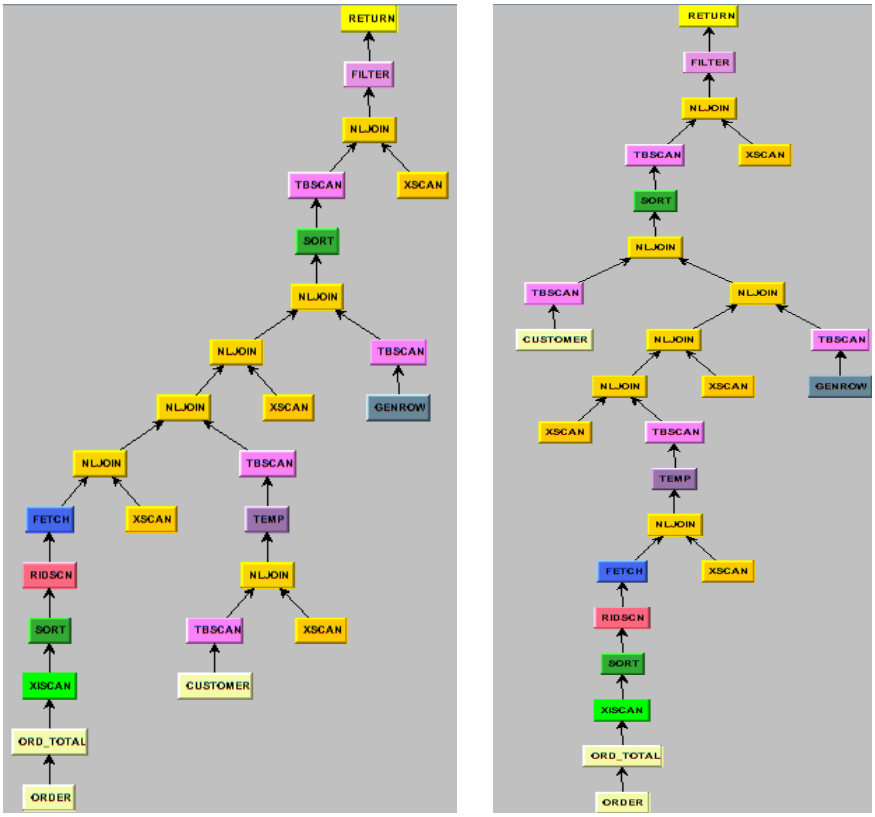
When plan diagram reduction with $\lambda = 20\%$ is applied, we obtain Fig. 6(c), where the cardinality is brought down to 21 from the original 42. Although there is an appreciable degree of reduction, it does not go to the extent of being "anorexic" (around 10 plans or less) – this is in marked contrast to the relational world, where anorexic reduction was invariably achieved with this λ setting [3]. More interestingly, anorexia could not be achieved even after substantial increases in the λ setting – in fact, only at the impractically large value of $\lambda = 150\%$ was this objective reached!

Complex Template. We now turn our attention to QTXB2, the complex XQuery template shown in Fig. 8. This template attempts to retrieve the names, number of items bought, and average discount provided for customers who live within a (parametrized)



(a) Plan diagram (b) Cost diagram (c) Reduced PD ($\lambda = 20\%$)

Fig. 6. Plan, cost and reduced plan diagrams for XBench QTXB1
(X-Axis: ORDER /order/total, Y-Axis: /customers/customer/address_id)



(a) Plan P1 (red) (b) Plan P2 (blue)

Fig. 7. Plan trees (QTXB1)

address value and whose total purchases are within a (parametrized) value, with the output sorted on the discount rates. QTXB2 is based on XQueries 9, 14 and 19 of XBench and incorporates most features provided by the XQuery language. Specifically, it has all the FLWOR clauses, expressions involving wild cards (`$ord//item_id`), and navigation on the sibling axis (`$ord//item_id/parent::order_line/discount_rate`) in the *return* clause. It also has predicates involving positional node access

(`$add/exists($add/street_address[2])`) in the *where* clause. Finally, aggregate functions provided by XQuery, such as *count* and *avg* are also employed.

```
for $cust in xmlcol (CUSTOMER.CUSTOMER)/customers/customer[address_id :varies]
let $add := xmlcol(ADDRESS.ADDRESS)/addresses/address[@id=$cust/address_id]
let $order in xmlcol(ORDER.ORDER)/order[total :varies and customer_id=$cust/@id]
 where exists($ord) and $add/exists($add/street_address[2])
 order by $cust/discount_rate
 return <Customer> {$cust/user_name} {$cust/discount_rate} <NoOfItems>
  {count($ord//item_id)} </NoOfItems> <AvgDiscount>
  {avg($ord//item_id/parent::order_line/discount_rate)} </AvgDiscount> </Customer>
```

Fig. 8. XQuery template for XBench (QTXB2)

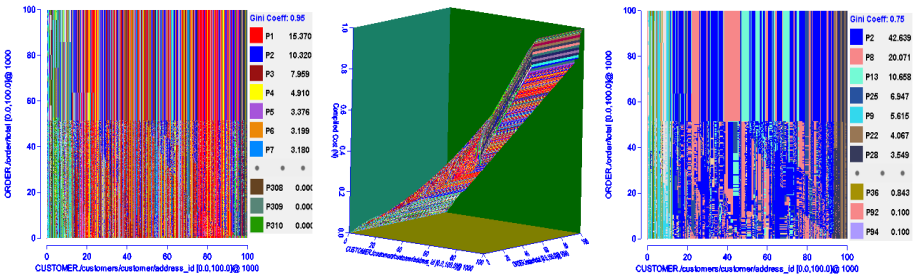
The plan diagram produced with QTXB2 is shown in Fig. 9(a), produced at a resolution of 1000*1000. We observe here that there are as many as 310 different plans! With a cursory glance, it would almost seem as though a different plan is chosen for each selectivity value. Further, the spatial layouts of these plans are extremely complex, throughout the selectivity range. Finally, the Gini skew co-efficient has a very high value of 0.95 indicating that a miniscule number of plans occupy virtually all of the area. In contrast to the basic template, QTXB1, even when the differences that arise due to parameter switches are disregarded, we are still left with 222 structurally different plans.

Drilling down into these plan structures, we find that the differences arise due to many factors, including changes in access methods and join orders, as well as ancillary operators such as SORT. However, variations in the position of application of the structural and value-based predicates result in the largest number of differences. These differences manifest themselves in the positioning of the NLJOIN-XSCAN operator pairs, where both types of predicates are applied.

An important point to note in Fig. 8 is that the structural volatility of plans in the diagram space is extremely high – that is, neighboring plans, even in adjacent parallel lines, are structurally very dissimilar and incorporate most of the differences discussed above. Another interesting observation is that the SORT operator corresponding to the *order by* clause is not always deferred to the end – this is in contrast to SQL query plans, where such sorts are typically found in the terminal stem of the plan tree. In fact, here, it is even found at the leaves of some plans!

The associated cost diagram is shown in Figure 9(b), where we see that along the `/order/total` VSX axis, the cost steadily increases with selectivity until 50%, and then saturates. Along the `/customers/customer/address_id` VSX axis, however, the cost monotonically increases with the selectivity throughout the range.

Finally, when reduction at $\lambda = 20\%$ is applied to the plan diagram, Fig. 9(c) is obtained wherein 13 plans are retained. The geometries of these surviving plans continue to be intricate even after reduction, whereas in the relational world, cardinality reduction was usually accompanied by a corresponding simplification in the geometric layout as well.



(a) Plan diagram (b) Cost diagram (c) Reduced PD (λ = 20%)

Fig. 9. Plan, cost and reduced plan diagrams for XBench QTXB2
(X-Axis: /customers/customer/address_id, Y-Axis: /order/total)

4.3 Plan Diagrams with TPoX

```
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
for $cust in xmlcol(CUSTACC.CADOC)/c:Customer/c:Accounts/c:Account[c:Balance/c:WorkingBalance :varies]
for $ord in xmlcol(ORDER.ODOC)/FIXML/Order[@Acct=$cust/@id/fn:string(.) and OrdQty/@Cash :varies]
 order by $cust/../../c:Name/c:LastName/text()
 return <Customer> {$cust/../../c:Name/c:LastName} {$cust/../../c:Nationality} </Customer>
```

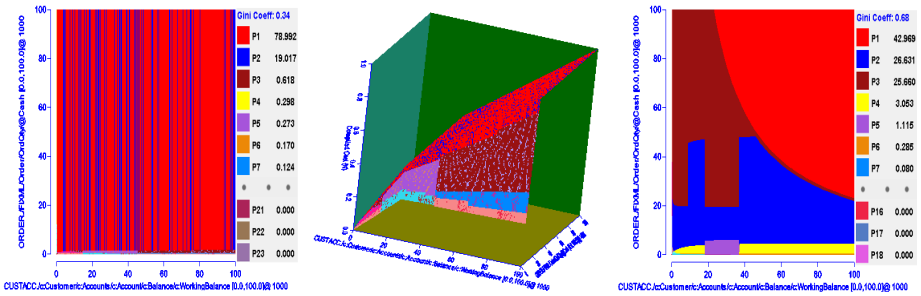
Fig. 10. XQuery template for TPoX (QTX_SEC)

Here, we present the results obtained with an XQuery template on the TPoX XML transaction processing benchmark. The query template, which is shown in Fig. 10, is based on the cust_sold_security.xqr XQuery given as part of the benchmark. This template, which we will hereafter refer to as QTX_SEC, returns details of customers with account(s) whose working balance(s) are within a (parametrized) value, and have traded one or more securities, with the trading amounts of the associated orders being within a (parametrized) value – the final result is alphabetically sorted on customer last names.

The plan diagram (at a resolution of 1000*1000) for QTX_SEC is shown in Fig. 11(a). It consists of 23 plans with plan P1 occupying about three-quarters of the space and plan P23 present in less than 0.001% of the diagram, resulting in an overall Gini coefficient of 0.34. Further, the plan cardinality decreases to 10, when differences between plan trees due to parameter switches, such as PREFETCH and SORT, are not considered. We also see that the plan diagram predominantly consists of vertical blue bands (plan P2) on the red region (plan P1). Only close to the X and Y-axes do we find other plans, such as the yellow vertical stripe of plan P4, and the brown and purple horizontal bands of plans P3 and P5, respectively.

The associated cost diagram is shown in Fig. 11(b), where we observe a strongly non-linear behavior with respect to the VSX selectivities. When the plan diagram is reduced with λ = 20%, the number of plans comes down to 11 plans, with the blue bands (P2) being swallowed by the red plan (P1).

The TPoX benchmark also features a semantically equivalent SQL/XML version of the XQuery example used above. We converted this version into an equivalent template, for which we obtained the plan diagram shown in Fig. 11(c). The operators found in the plans of the SQL/XML plan diagram are the same as those found in the XQuery plan diagram. However, the choice of optimal plans vastly differs across the diagrams – note



(a) Plan diagram (b) Cost diagram (c) SQL/XML plan diagram

Fig. 11. Plan and cost diagrams for TPoX QTX_SEC; Equivalent SQL/XML plan diagram
(X-Axis: /c:Customer/c:Accounts/c:Account/c:Balance/c:WorkingBalance, Y-Axis: /FIXML/Order/OrdQty/@Cash)

that the SQL/XML plan diagram throws up only 18 plans, and that too, without any striking patterns.

Further, although the end query results are identical, there are substantial cost variations between the two query template versions. Specifically, the SQL/XML template has minimum and maximum costs of 2.77E3 and 1.14E7, respectively, whereas the XQuery template has minimum and maximum costs of 2.76E3 and 1.65E6, respectively. These differences in costs are due to the application of the SORT operator at different positions in the plan trees. Sorting is always carried out at the stem of the plan tree, after applying all predicates, in the case of SQL/XML, whereas the application of sorting is sometimes done earlier in the case of XQuery. For the cases where sorting is applied at the stem level for both types of queries, there is also a difference in the estimated cardinalities, which is reflected in the cost estimates. This indicates that, even when the underlying data and the query semantics are the same, the specific choice of query interface may have a material impact on the runtime performance.

4.4 Plan Diagrams with TPCH_X

```
for $c in xmlcol(CUSTOMERS.CUSTOMERS)
 /Customers/Customer/[AcctBal :varies]
let $n:=xmlcol(NATIONS.NATIONS)/Nations/Nation[@key=$c/NationKey]
let $o := xmlcol(ORDERS.ORDERS)
 /Orders/Order[$c/@key=CustKey and TotalPrice :varies]
 where fn:exists($o)
 order by $c/MktSegment
 return <Customer> {$c/Name} {$n/Name} {$c/MktSegment}
  {$o/OrderKey} </Customer>
```

(a) XQuery template

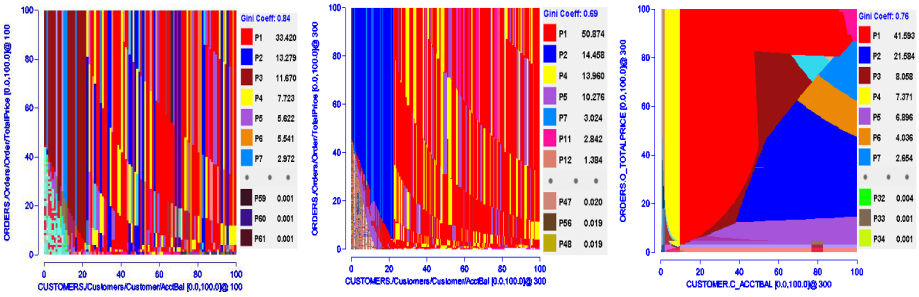
```
select c_name, n_name,
 c_mktsegment, o_orderkey
from customer, nation, orders
where c_acctbal :varies and
  c_natkey=n_natkey and
  o_custkey=c_custkey
  and o_totalprice :varies
order by c_mktsegment
```

(b) SQL query template

Fig. 12. XQuery and SQL templates for TPCH_X/TPCH

An XQuery template based on Query 10 of TPCH_X is shown in Fig. 12(a). This template retrieves the names, home nations, and marketing segments of customers, and the identifiers of all their purchases, with the results ordered on the marketing segments. The plan diagram for this template is shown in Fig. 13(a), where we again observe an extremely complex diagram populated with 61 different plans, appearing mostly

as rapidly alternating bands of colors. When this diagram is subject to reduction with $\lambda = 20\%$, we obtain Fig. 13(b), which retains 19 plans and is therefore not anorexic in nature. In fact, λ had to be increased to as much as 50% to obtain an anorexic diagram.



(a) Plan diagram (b) Reduced PD (λ = 20%) (c) SQL plan diagram

Fig. 13. Plan and reduced plan diagrams for TPCH_X and plan diagram for TPCH
(X-Axis: /Customers/Customer/AcctBal, Y-Axis: /Orders/Order/TotalPrice)

As a matter of curiosity, we also investigated the behavior of the equivalent (in terms of the result set) SQL query template, shown in Fig. 12(b). The associated plan diagram in Fig. 13(c) throws up a much simpler picture, both in terms of cardinality (34 plans) and in the spatial layouts of the optimality regions. Further, the estimated execution costs for the XQuery template are *orders of magnitude* higher in comparison to those obtained with the SQL template! Assuming that the optimizer's modeling quality is similar in both environments, these results indicate that database administrators of hybrid systems must make a careful choice of data representation to provide the best performance for their users.

4.5 General Observations

During the course of our experimentation on XQuery plan diagrams, a few general observations emerged, which are highlighted below:

- The presence of an *order by* clause in the XQuery templates results in a dramatic increase in the richness of plan diagrams, with respect to both the density and geometric complexity. The reason is as follows: XML is inherently ordered, and results are always produced in document order (without the presence of *order by*). With the presence of *order by* on a path, a low-cost sort can potentially be accomplished at several steps in the optimization process, and hence there is a large set of similarly-costed alternative plans to choose from, many of which surface as the locally-optimal plan at one or the other location in the selectivity space.
- The complexity of the plan diagrams increases with the complexity of the predicates involved in the XQuery template, with advanced features such as navigation on different axes (sibling and parent), wild cards and positional node access triggering this behavior.
- The position of predicates – whether appearing in the XPath expression or in the *where* clause of the XQuery templates – has a significant impact on the complexity

of plan diagrams in terms of both plan cardinality and spatial distribution. Further, and very importantly, this shift of position also results in plans with substantially changed costs. Ideally, in a truly declarative world, all equivalent queries should result in the optimizer producing the same plan – however, we see here that XOpt is not able to automatically sniff out these important rewriting opportunities.

Taken in toto, the above results seem to suggest that considerable scope remains for improving on the construction of current hybrid optimizers.

5 Related Work

To the best of our knowledge, there has been no prior work on the analysis of *industrial-strength* native XQuery optimizers using the plan diagram concept. The closest related effort is the plan-diagram-based study of SUCXENT [2], an XML processing system that uses Microsoft SQL Server as the backend relational storage engine. They studied the behavior of this optimizer in the context of XPath processing, by first converting all XPath queries to their equivalent SQL versions.

An XML plan diagram *instance* was also shown in [11], using IBM DB2, to motivate the need for accurate cardinality estimations of XQuery expressions – in their experimental setup, the optimal plan choice is highly volatile, varying with small changes in selectivity, and inaccurate estimations of cardinalities result in choosing plans that are worse than the optimal by orders of magnitude.

A flexible optimization framework, incorporating both rules and cost estimates, was presented in [12] for visualizing the XQuery optimization process in a native XML DBMS environment. The framework supports implementing, evaluating, and reconfiguring optimization techniques, even during run-time, through a visual tool. However, all these features are applicable to individual queries, and are not intended for visualization over a parameter space.

Finally, an evaluation of open-source XQuery processors was carried out in [6], but their focus was on characterizing the response time performance for specific benchmark queries.

6 Conclusions and Future Work

In this paper, we have attempted to analyze the behavior of XOpt, an industrial-strength XQuery/SQL hybrid optimizer, using the concept of plan diagrams proposed some years ago in [9]. We first addressed the issue of what comprises a XML selectivity space, and the mechanisms to be used to bring about the desired selectivity variations. Then, we enumerated the constraints that need to be satisfied in formulating XQuery templates so as to obtain meaningful plan diagrams. Subsequently, we provided a detailed report on XOpt's plan diagram behavior over a representative set of complex XQuery templates constructed over popular XML benchmarks.

Our experimental results indicate that XML plan diagrams are significantly more complex than their relational counterparts in terms of plan cardinalities, densities and spatial layouts. In particular, we observe a pronounced "banding" effect resulting in

wave-like patterns. Further, the presence of even syntactic expressions, such as *order by*, visibly increase the complexity of the resulting diagrams. We also find that these diagrams are not always amenable to anorexic reduction at the 20% cost-increase threshold found sufficient in the relational literature, often requiring substantially higher thresholds to achieve the same goal. Another interesting facet is that equivalent XML and SQL queries typically produce substantially different cost estimations from the optimizer. Overall, these results suggest that important challenges remain to be addressed in the development of effective hybrid optimizers.

While our analysis was restricted to XOpt in this study, it would be interesting to profile the plan diagram behavior of other industrial-strength XML query processing engines as well. Further, going beyond the two-dimensional query templates evaluated here to higher dimensions will provide greater insight into addressing the design issues underlying these systems.

References

1. D. Barbosa, A. Mendelzon, J. Keenleyside and K. Lyons, "ToXgene: An extensible template-based data generator for XML", *Proc. of 5th Intl. Workshop on the Web and Databases (WebDB 2002)*, June 2002.
2. S. Bhowmick, E. Leonardi and H. Sun, "Efficient Evaluation of High-Selective XML Twig Patterns with Parent Child Edges in Tree-Unaware RDBMS", *Proc. of 16th ACM Conf. on Information and Knowledge Management (CIKM)*, November 2007.
3. Harish D., P. Darera and J. Haritsa, "On the Production of Anorexic Plan Diagrams", *Proc. of 33rd Intl. Conf. on Very Large Data Bases (VLDB)*, September 2007.
4. A. Eisenberg and J. Melton, "SQL/XML and the SQLX Informal Group of Companies", *SIGMOD Record (30)*, September 2001.
5. A. Hulgeri and S. Sudarshan, "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions", *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2002.
6. S. Manegold, "An empirical evaluation of XQuery processors", *Information Systems Journal*, April 2008.
7. M. Nicola, I. Kogan and B. Schiefer, "An XML Transaction Processing Benchmark (TPoX)", *Proc. of 2007 ACM SIGMOD Intl. Conf. on Management of Data*, June 2007.
8. M. Nicola and B. Linden, "Native XML support in DB2 universal database", *Proc. of 31th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2005.
9. N. Reddy and J. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers", *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB)*, August 2005.
10. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt and J. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities", *Proc. of 25th Intl. Conf. on Very Large Data Bases (VLDB)*, September 1999.
11. J. Teubner, T. Grust, S. Maneth and S. Sakr, "Dependable Cardinality Forecasts for XQuery", *Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2008.
12. A. Weiner, T. Härder, and Renato O. Silva, "Visualizing Cost-Based XQuery Optimization", *Proc. of 26th Intl. Conf. on Data Engineering (ICDE)*, March 2010.
13. B. Yao, M. Ozsu and N. Khandelwal, "XBench Benchmark and Performance Testing of XML DBMSs", *Proc. of 20th Intl. Conf. on Data Engineering (ICDE)*, March 2004.
14. `http://docs.oracle.com/cd/B14117_01/appdev.101/b10790/xdb01int.htm`
15. `http://msdn.microsoft.com/en-us/library/ms345117(v=sql.90).aspx`
16. `http://www.tpc.org/tpch`