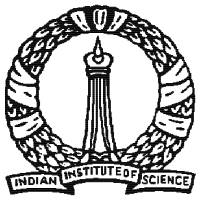
Exploring the Semantics of Plan Diagrams

A Thesis
Submitted for the Degree of
Master of Science (Engineering)
in the Faculty of Engineering

By
Bruhathi HS



Supercomputer Education and Research Centre
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012, INDIA

February 2012

Abstract

Database Management Systems (DBMS) are popular software packages for storing and managing enterprise data. Modern DBMS typically store data in the form of tables, called relations, and query the information using the Structured Query Language (SQL), which is declarative in nature. The number of candidate strategies, called "plans", by which the query can be processed, is exponential in the number of relations. Database systems therefore incorporate a "query optimizer" module, which explores this exponential space using dynamic programming techniques, to find the best query execution strategy. Query optimizers are especially critical for efficiently processing the complex queries that characterize current decision-support applications.

In recent times, a fresh perspective on the behavior of query optimizers has been introduced through the concept of "plan diagrams". A plan diagram is a color-coded pictorial enumeration of the optimizer's plan choices over a query parameter space. In this thesis, we investigate a variety of issues related to the semantics of plan diagrams, covering the structural characteristics of parametric-optimal plans, the diagram coloring paradigm and the underlying database model. Our specific contributions are the following:

The join order of an execution plan tree of a query is the order in which its relations are joined and finding the optimal join order is one of the most crucial tasks in query optimization, with any savings in this front being extremely beneficial. In this thesis, as our first contribution, we develop and discuss a *structure-based* reduction scheme as a potent alternative to the popular cost-based reduction schemes that have been explored in the past. In this new reduction scheme, we merge all plans that have identical join orders into a single entity, leaving us with a diagram that has as many colors as there are unique join orders. We discuss join order caching as an immediate application of structure-based reduction that provides the "best" plan at every

point along with savings in query optimization time, when the join order cardinality is low or moderate. While experimental results indicate that the resulting join order cardinalities are low or moderate in many cases, we also encounter situations wherein the cardinality is high. For the latter case, we present the SRE (Small Relation Elimination) heuristic, that removes small sized relations in the join orders to bring down the join order cardinality. Even though the plan optimality is no longer guaranteed now, experimental evaluation of the heuristic shows that the cost-increase of any query point is within acceptable limits. Finally, we study the occurrence of common join orders in the plan diagrams produced in two different engines. Experimental results show that this intersection is surprisingly very sparse.

Structure-based reduction provides an overview of the differences between plans in a plan diagram. As our second contribution, we investigate a deeper comparison between plans that is based on their complete plan tree structures. Specifically, we *semantically* color the plan diagrams such that the differences in colors between any pair of plans, reflects the differences in their structures. With this approach, the plan diagram intrinsically reflects the diversity in the parametric-optimal space. The challenges here include designing a quantitative metric for assessing plan differences, and developing transformation techniques for accurately representing these differences in the three-dimensional color model. In particular, we adapt Kruskal's Iterative Steepest Descent multidimensional scaling technique, and test its representational quality through coloring a rich diversity of plan diagrams on benchmark database environments over a suite of commercial database engines. Our experimental results indicate that the vast majority of plan distances can be represented with satisfactory visual accuracy. Given this, we found that, in most of the plan diagrams, more than half the space is colored with shades of the same color, implying that large areas of plan diagrams are occupied by plan trees that are structurally similar.

As our last contribution, we reengineer the plan diagram notion to the flexibly structured world of eXtensible Markup Language (XML), which is the de facto standard for information transfer on the Internet. Since XML queries are founded on monadic second-order logic, as opposed to the first order logic of relational DBMS, they possess significantly more expressive power than their relational counterparts, in the process exacerbating the query optimization

challenge. We analyze, through the production of XML plan diagrams, the behavior of a hybrid commercial optimizer that optimizes both SQL and XML queries in a unified framework. Our experiments cover a variety of XML database benchmarks, including TPoX, XBench and TPCH_X (XML version of the TPC-H benchmark), using a set of complex XQuery templates. The results indicate that XML plan diagrams often feature complicated plan geometries that remain in the plan diagram even after reduction. In fact, for both TPoX and TPCH_X, an extremely high cost-increase threshold is required in order to remove the complex plan spatial layouts from the plan diagram. Also, we found that more than three-quarters of the plans were structurally very similar, implying that the optimizer makes extremely fine grained choices with small changes in input parameters.

All the three semantic features discussed above have been incorporated in the publicly-available Picasso optimizer visualization tool.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern DBMS (Database Management Systems) typically model data in the form of tables, called relations, and query the information using the Structured Query Languages (SQL). SQL is declarative in nature and hence specifies *what* has to be done, and not *how* to do it. An example SQL query is listed in Figure 1.1. This SQL query lists the mode of shipping for all items whose quantity is less than or equal to 26, and forms part of an order of price 21394 or less.

```
select l_shipmode
from orders, lineitem
where o_orderkey = l_orderkey
        and o_totalprice ≤ 21934
        and l_quantity ≤ 26
group by l_shipmode
order by l_shipmode
```

Figure 1.1: Sample SQL Query

The number of candidate strategies, called "plans" by which the query can be processed is exponential in the number of relations. The quality of a plan is measured in terms of its "cost", usually an estimate of its expected response time. Since the difference in cost between the best execution plan and a randomly chosen one can be very high, database systems incorporate a

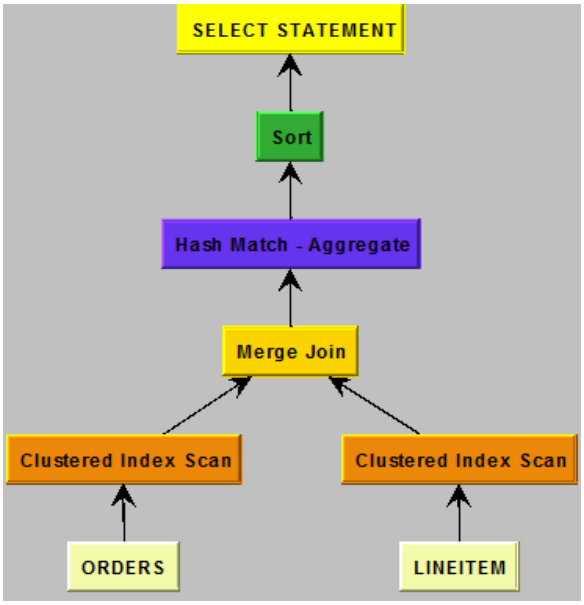


Figure 1.2: Sample SQL Query Execution Plan

"query optimizer" module to explicitly and automatically find the best query execution strategy. This objective is achieved by exhaustively exploring the exponential search space using dynamic programming techniques.

The best query execution plan as chosen by an optimizer for the SQL query listed in Figure 1.1, is shown in Figure 1.2. The query execution plan for this query uses the *Merge Join* algorithm to join the relations ORDERS and LINEITEM, and these relations are accessed through *Clustered Index Scan*.

The importance of query optimizers has elevated in recent times due to the high degree of complexity in queries, operating on large data stores, typically found in data warehousing applications. This is evident in benchmarks like TPC-H [62].

1.1 Query Templates

The cost of a given query execution plan is a function of many parameters, including the database structure and contents, the engine settings, the system configuration, etc. For a query on a given database and system configuration, the optimizer's plan choice is primarily a function of the *selectivities* of the base relations participating in the query – that is, the estimated number

```
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
```

Figure 1.3: Example SQL Query Template (QT8)

of rows of each relation relevant to producing the final result. Varying the selectivities of one or more of the base relations produces the selectivity space w.r.t. these relations. A "parameterized query template" is a query with additional predicates that produce queries across this selectivity space.

For example, consider QT8, the parameterized 2-D query template shown in Figure 1.3, based on Query 8 of the TPC-H benchmark. This query determines the market share of a nation within a given region for a given part type when the s_acctbal of the SUPPLIER relation and the l_extendedprice of the LINEITEM relation are within their respective parametrized values. Here, selectivity variations on the SUPPLIER and LINEITEM relations are specified through the s_acctbal :varies[1] and l_ extendedprice :varies predicates, respectively.

1.2 Optimizer Diagrams

In recent times, a fresh perspective on the behavior of query optimizers has been introduced through the concept of "optimizer diagrams" [38, 10, 14]. Primary among them is the *plan diagram*, which is a visual representation of the execution plan choices made by the optimizer

[1]To implement *:varies*, we use one-sided range predicates of the form Relation.attribute <= constant

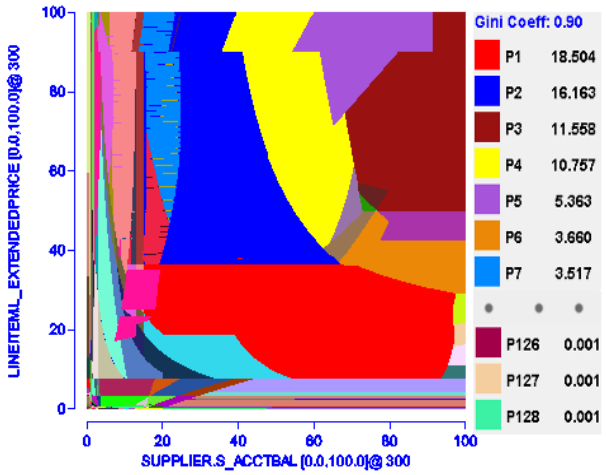
over an input parameter space, whose dimensions may include database, query and system-related features. In a nutshell, plan diagrams visually capture the geometries of the optimality regions of the parametric optimal set of plans (POSP) [23]. Currently, we support parameter spaces whose dimensions are comprised of *relational selectivities*.

The plan diagram for QT8 (produced with the Picasso [52] optimizer visualization tool on a popular commercial database engine) is shown in Figure 1.4(a). In this picture[2], each colored region represents a specific plan, and a set of 128 different optimal plans (from the optimizer's perspective), P1 through P128, cover the selectivity space. The value associated with each plan in the legend indicates the percentage area covered by that plan in the diagram – the biggest, P1, for example, covers 18.59% of the region, whereas the smallest, P128, is chosen in only 0.001% of the space. Just above the plan legend we display the Gini coefficient, which measures the *skew* in the volumes of the regions covered by the various plans featuring in the plan and reduced plan diagrams. It is a number between 0 and 1, where 0 corresponds to no skew (i.e. all plans cover approximately the same area) and 1 corresponds to extreme skew (i.e. one plan covers almost the entire space).

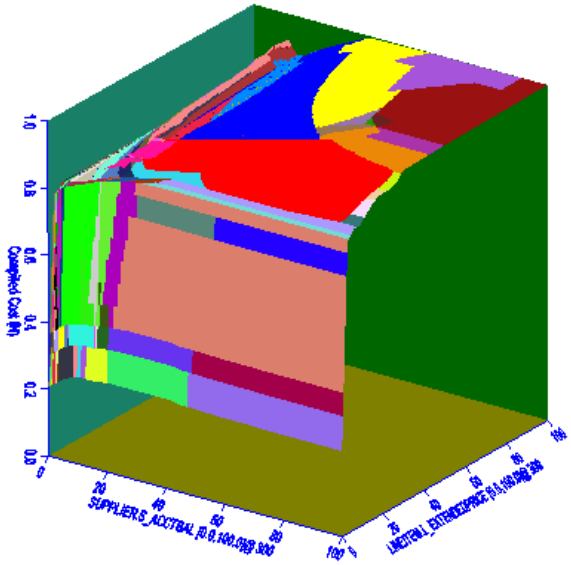
The mode of plan diagram production is the following [52]: Given a d-dimensional query template and a plot resolution of r, r^d queries that are uniformly distributed over the selected selectivity space are generated. Then, for each of these query locations, based on the associated selectivity values, a query with the appropriate constants is instantiated – the constants are determined by carrying out an "inverse-transform" of the statistical meta-data available from the optimizer, typically in the form of histograms. This query is then submitted to the query optimizer to be "explained", that is, to have its optimal plan computed and returned.

After the plans corresponding to all the query points are obtained, a different color is associated with each unique plan, and each point is colored with its associated plan's color. Then, the rest of the diagram is filled in by painting the hyper-rectangle around each point with the same color. For example, in a 2-D plan diagram with a uniform grid resolution of 10, there are 100 real query points, and around each such point a square of dimension 10x10 is painted with the point's associated plan color. Plans are colored based on a pre-defined coloring scheme, with

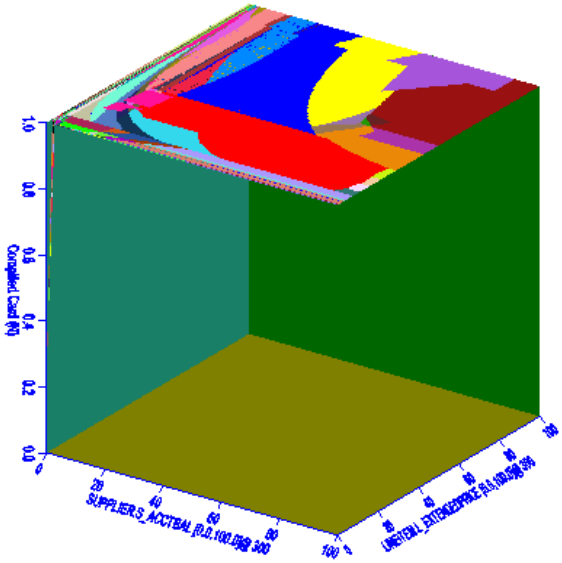
[2]The figures in this thesis should ideally be viewed from a color copy, as the grayscale version may not clearly register the features.



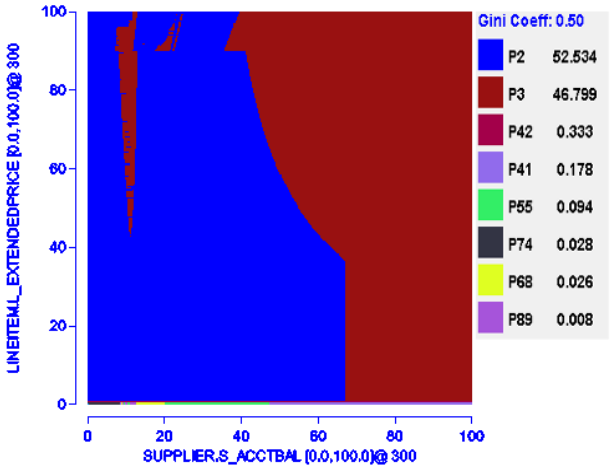
(a) **Plan Diagram**



(b) **Cost Diagram**



(c) **Card Diagram**



(d) **Reduced Plan Diagram (λ = 20%)**

Figure 1.4: Optimizer Diagrams for QT8 (X-Axis: SUPPLIER S_ACCTBAL, Y-Axis: LINEITEM L_EXTENDEDPRICE)

the biggest sized plan (size is measured in terms of volume coverage in the n-D space) being colored red, the second biggest being colored blue and so on.

In conjunction with the plan diagram, we have two additional diagrams, the *Cost Diagram* and the *Card Diagram*. The cost diagram quantitatively depicts the optimizer's (estimated) query processing costs of the plans shown in the plan diagram, while the cardinality diagram displays the optimizer's (estimated) query result cardinalities. The cost and card diagrams for the QT8 example are shown in Figures 1.4(b) and 1.4(c), respectively.

Plan diagrams are often found to have high plan cardinalities, with many plans occupying complex and non-convex boundaries. This can be observed in the plan diagram of Figure 1.4(a), which features a lot of complex plans with also a large skew in the areas of the regions covered by the various plans. This skew is captured by the 0.90 value of the Gini coefficient.

Given this complexity in plan diagrams, it is natural to investigate the possibility of making the plan diagrams simpler in terms of their plan cardinalities, that is, to obtain a *reduced plan diagram*. Recently, a cost-based approach to the reduction problem has been explored [10], where given a plan diagram and a cost-increase-threshold (λ) specified by the user, the plan diagram is recolored to a simpler picture, featuring only a subset of the original plans, such that the cost of *any* recolored query point does *not* increase by more than λ percent, relative to its original cost. That is, some of the original plans are "completely swallowed" by their siblings, leading to a reduced number of plans in the diagram, without materially affecting the query processing quality. Experimental evaluation with this approach has shown that the complex plan diagrams are often reduced to be "anorexic" (small absolute number of plans) with only a marginal increase in query processing costs. For example, the QT8 plan diagram (Figure 1.4(a)) can be reduced with λ =20% to the diagram shown in Figure 1.4(d), where only *eight* of the original 128 plans are retained.

1.3 Structure-based Reduction of Plan Diagrams

As the first contribution in this thesis, we present and analyze a different plan diagram reduction scheme that is based on the *structure* of plans present in the diagram, rather than cost. The struc-

ture of a plan is largely determined by its *join order*, which represents the sequence in which its base relations are joined, and its determination is considered to be the most challenging task in optimizing a query. So, in our structure-based reduction scheme, we coalesce all plans that have identical join orders into a single entity. Therefore, after the reduction process, there are as many colors in the diagram as the number of distinct join orders.

Structure-based reduction has applications in join order caching of plans when the join order cardinality is low or moderate. Here, when a query located in the plan diagram is issued, the query is not optimized from scratch. Instead, the join orders are used as plan skeletons, using which complete plans are fleshed out and evaluated to find the *best plan*, thus saving considerable computational effort. An attractive feature of the structure-based reduction scheme is that it makes no assumptions on the underlying cost model, and hence is applicable to all kinds of queries – unlike cost-based reduction where PCM (Plan Cost Monotonicity) is required to hold [10]. Yet another advantage is that it functions solely through the standard "hint" mechanism provided by optimizers, and does not require advanced features such as Foreign Plan Costing (FPC) [10] – this makes it applicable in generic database engines.

Experimental evaluation on plan diagrams obtained from two industrial-strength commercial optimizers, one being top-down and the other a bottom-up optimizer, in benchmark environments indicate that the number of distinct join orders is typically in the range of 10 to 60. This is true even for dense plan diagrams that have hundreds of plans. Since fresh optimization of a query typically takes considerably more time (by about a factor of 20) than constructing its best plan from the given join order, it is evident that join order caching is desirable for plan diagrams whose join order cardinality is low. Moreover, even the factor of 20 is conservative due to including overheads such as query validation and re-writing – if join order caching were to be organically internalized in the optimizers, the expansion time is likely to come down even further, making it also viable for plan diagrams with medium-scale join order cardinalities.

Further, if faced with high join order cardinalities, we could take recourse to the SRE (Small Relation Elimination) heuristic-based solution: We remove from the join orders obtained, the relatively smaller sized relations, thereby obtaining a pruned set of partial join orders. This step is based on the hypothesis that the number of join orders is large because of the variations in the

joins of these smaller relations and that the change in the join order with respect to them does not have a huge impact on the join order cost. For each unique partial join order, we assign a representative complete join order, chosen randomly from among the complete join orders that are built from the partial join order. When a new query has to be optimized, we evaluate the representative join orders corresponding to each unique partial join order and choose the one yielding the least cost.

We evaluated the above heuristic experimentally on some sample complex queries, and found that after applying the heuristic, the number of join orders in the reduced diagram came down to around 20 in most cases, with the per-query cost increase less than 30% over most of the diagram. Further, even for the remaining queries, the cost-increase was less than 100%, generally considered to be the acceptable deviation from optimality, in the database community. Thus, structure-based reduction provides a potent mechanism to implement join order caching for complex queries, either directly or through the heuristic.

Finally, we study the number of common join orders present in the plan diagrams of the two engines. Surprisingly, we found the intersection to be *null* in most cases. This can be because of various reasons including different cost models, variations in the implementation of different join algorithms and employment of different heuristics to prune the search space.

1.4 Semantic Coloring of Plan Diagrams

Structure-based reduction scheme provides a bird's eye view of the differences found between plans in a plan diagram. In order to determine how different any two specific plans are, in terms of their join orders or more generally, their complete plan tree structures, we need to drill down into the plans. This process is cumbersome when more than a handful of plans have to be analyzed. Our second contribution in this thesis alleviates the problem by *semantically* coloring plan diagrams as opposed to the current plan-agnostic coloring scheme, wherein the largest plan (in terms of volume) is assigned red, the second largest is assigned blue, and so on.

Specifically, we color the plan diagrams such that the differences in color between any pair of plans reflects the differences in their structures. For instance, if the biggest and second-

biggest plans are very similar, they would both be assigned close shades of the same color, say red. With this new approach to coloring, the plan diagram itself provides a first-cut reflection of the plan-tree differences without having to go through the details of every plan. To assign differences to every pair of plans, we adopt the plan differencing strategy from [14, 52], where quantification of plan differences is done using the Jaccard distance metric, making all plan differences to be in the interval $(0, 1]$. Given this framework, the modelling of the problem is as follows:

Problem Statement: Consider an undirected complete graph $G(V, E)$ with m vertices, where each vertex $v_i \in V$ represents a POSP (Parametric Optimal Set of Plans) plan. A vertex v_i has an associated weight w_{v_i}, representing the fractional volume covered by the plan in the plan diagram. Each edge $e_{ij} \in E$ has an associated weight p_{ij} representing the structural distance between the plans at its vertices v_i and v_j. Also, each vertex pair (v_i, v_j) is assigned a weight w_{ij}. The objective now is to come up with an efficient algorithm that assigns a unique color to all the vertices in V such that, given any pair of vertices (v_i, v_j), their color distance c_{ij} is as close as possible to the weight p_{ij} of the edge joining them. Further, if a choice has to be made, vertex pairs that are assigned higher weights (w_{ij}) receive preferential treatment in obtaining a more accurate coloring for their distances. That is, our aim is to minimize (as close to 0 as possible) the following objective function:

$$\sum_{v_i=1}^{m-1} \sum_{v_j=v_i+1}^{m} w_{ij}(p_{ij} - c_{ij})^2$$

A variety of assignments are possible for the weights w_{ij}: For example, they can be $w_{v_i} + w_{v_j}$, giving preference to plan pairs with larger total area; or 1, which treats all vertex pairs equally; or, $\dfrac{1}{w_{v_i} + w_{v_j}}$, where small sized plans are preferentially colored better.

We choose the RGB range as our color space, hence constraining the output space to a cube. So, in order to utilize the output space completely, we scale all the plan-pair differences by the length of the cube's diagonal. It is important to note that an ideal solution which gives an

objective value of zero may not always exist, since scaling down distances to lower dimensions is an inherently lossy procedure.

As our optimization strategy, we consider methods from the family of multidimensional scaling (MDS) techniques. We first evaluate the metric SMACOF [7] (Scaling by Majorizing a Complicated function) approach, but the output of this iterative algorithm is in $\mathbb{R}^3$ and scaling down the values into the color cube leads to material deterioration in the quality of coloring. However, through a quick analysis we observe that the non-metric SD (Steepest Descent) algorithm [29, 30], can be easily modified to accommodate the cube constraint and hence adapted it to suit our requirements. A downside of the adaption is that the convergence rate guarantees provided by classical SD may no longer hold, but our empirical experience is that even for plan diagrams with hundreds of plans, convergence is achieved within a few minutes.

We test the representational quality of our SD-based technique by coloring a rich diversity of plan diagrams produced from benchmark database environments. Our results indicate that sufficiently low objective function values (< 0.1) were always obtained, with the vast majority of the plan-pairs being colored well (ratio of color distances to their plan distances is within 30% of the ideal). This indicates that most of the plan diagram is represented with satisfactory visual accuracy. Further, these results closely match those from the unconstrained SMACOF algorithm with respect to the average case behavior.

When we analyzed the semantically colored diagrams, we found that typically more than half the space is colored with shades of the same color, implying that large areas of plan diagrams are occupied by structurally similar plan trees. Also, we found that these similar plans generally occupy interior regions, away from the axes.

1.5 XML Plan Diagrams

In recent years, data represented in the hierarchically-structured XML document format has found its way into mainstream database technology. Querying of this XML data is largely done through an XML query language called XQuery [66]. An example XQuery taken from the DCMD (Data Centric Multiple Documents) workload of the XBench benchmark [51] is shown

```
XQUERY
for $a in db2-fn:xmlcolumn("ORDER.ORDER")/order
        where $a/total <= 975
        return
          <Output>
            {$a/@id}
            {$a/order_date}
            {$a/ship_type}
          </Output>
```

Figure 1.5: Example XQuery

in the Figure 1.5. The XQuery has been modified to conform to the XQuery optimizer's syntax. This query lists the orders (order id, order date and ship date), with total amount larger than or equal to a certain number (975.0).

The optimal plan for the XQuery of Figure 1.5 chosen by the XQuery optimizer is shown in Figure 1.6. This plan accesses the ORDER relation using the XML index on the order_total attribute through the *XISCAN* operator. It then uses the *XSCAN* operator to navigate the XML fragments, evaluate predicates and to extract the document fragments to provide the required set of XML elements in the result.

As a final contribution of this thesis, we re-engineer the plan diagram notion to the flexibly structured world of eXtensible Markup Language (XML), which is the de facto standard for information transfer on the Internet. Since XML queries are founded on monadic second-order logic, as opposed to the first order logic of relational DBMS, they possess significantly more expressive power than their relational counterparts, in the process exacerbating the query optimization challenge. We analyze, through the production of XML optimizer diagrams, the behavior of a XML query industrial-strength optimizer that optimizes XQuery [65], the W3C recommended standard for querying XML data. The challenges involved in obtaining the XML plan diagrams include identifying the validity constraints on XQuery templates and determining the granularity at which the statistics need to be collected (document level or node level) for selectivity computation. After obtaining the XML plan diagrams, we conduct experiments that cover a variety of XML database benchmarks, including TPoX [35], XBench [51] and TPCH_X

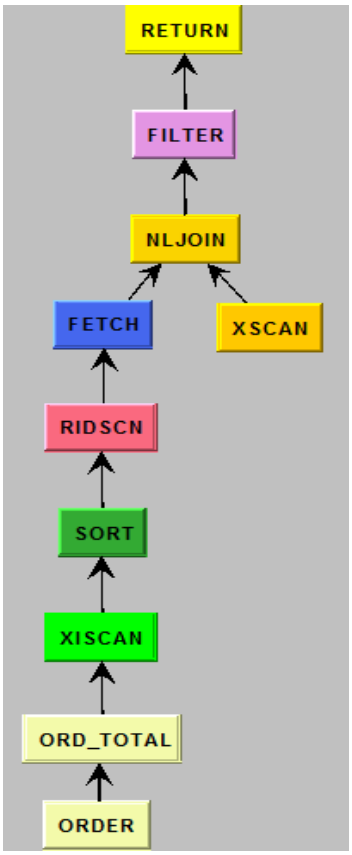


Figure 1.6: Example XQuery Execution Plan

(XML version of the TPC-H [62] benchmark), using a set of complex XQuery templates. The results indicate that XML plan diagrams often feature complicated plan geometries that remain in the plan diagram even after both cost and structure-based reductions. In fact, for both TPoX and TPCH_X, a very high cost-increase threshold is required in order to remove the complex plan geometries using cost-based reduction. With respect to structure-based reduction, however, we find that the number of final join orders is very sparse (in single digits), even when the query contained more than 5 relations. This fact along with the observation that more than half of the plans were structurally very similar to each other are reflected in the semantically colored versions of the XML plan diagrams.

1.6 Contributions

In this thesis, we explore a variety of semantic features in plan diagrams. We first investigate a *structure-based* reduction strategy as a complementary method to the cost based reduction schemes. In structure-based reduction, all plans that have identical join orders are merged into a single entity. As one of its potent applications, we propose join order caching for plan diagrams that is very useful when the join order cardinality is low or moderate. Join order caching provides the "best" plan for every query location along with savings in optimization time. Among the many pros of structure-based reduction in comparison to cost-based reduction strategies, we make no assumptions on the underlying cost model, and hence can apply structure-based reduction to all kinds of query templates. We present experimental results obtained by applying structure-based reduction on a suite of plan diagrams. While the results indicate that the resulting join order cardinalities are low or moderate in many cases, we also encounter situations wherein the cardinality may be high. For the latter case, we present the SRE (Small Relation Elimination) heuristic, that removes small sized relations in the join orders to bring down the join order cardinalities. Even though the plan optimality is no longer guaranteed, we present experimental results, which show that the cost-increase of any query point is within acceptable limits. Finally, we study the occurrence of common join orders in the plan diagrams produced in two different engines, for the same set of plan diagrams as used in structure-based reduction experiments. We surprisingly found that there are *no* common join orders between the plan diagrams in most cases.

While structure-based reduction provides a broad overview of the differences between plans present in plan diagrams, for analysis of differences between specific plans, it is necessary to manually look into the details of the various plans. As a second contribution in this thesis, we facilitate such analysis of plans by developing techniques to semantically color plan diagrams - to color plans in the plan diagrams such that the difference in color between any two plans reflects the differences in their plan structures. We first adopt the plan tree differencing methods from [52, 14], for assigning quantitative differences between any two plans. Then, we adapt *Kruskal's Iterative Steepest Descent* method, a multidimensional scaling (MDS) technique to solve our coloring problem. Through extensive experimentation on a host of plan diagrams

we found that the adapted technique represents the plan tree differences in color space with reasonable visual accuracy. Also, we found that large areas in the plan diagrams are colored with similar shades, implying that plans occupying larger areas have structurally similar plan trees.

As the last contribution of our thesis, we extend the plan diagram notion to the flexibly structured world of XML. We discuss the complexities involved in the process, which include determining the granularity at which the statistics have to be collected, and identifying the necessary conditions for a valid XQuery template. We provide experimental results obtained over a variety of benchmark environments. The benchmark environments include TPoX [35] and XBench [51], which are native XML benchmarks, as well as TPCH_X, our XML equivalent of the classical TPC-H [62] relational benchmark. With respect to results, we found that the XML plan diagrams produced, often feature patterns that merit further investigation and that there are considerable differences between the plan diagrams produced between equivalent relational and XML database environments. Finally, through structure-based reduction of XML plan diagrams, we show that the join order cardinalities of plan diagrams are low, and by semantically coloring these XML plan diagrams, we show that many plans are often structurally very similar.

All the above-mentioned semantic features have been incorporated into the publicly-available Picasso optimizer visualization tool [52].

1.7 Organization

The remainder of this thesis is organized as follows: Chapter 2 provides an overview of related research. Structure-based reduction is elaborated in Chapter 3, followed by a detailed description of the techniques developed for semantic coloring of plan diagrams in Chapter 4. Further, we discuss the methods and complexities involved in obtaining XML plan diagrams in Chapter 5. Finally, we conclude in Chapter 6 with a discussion of promising avenues for further investigation.

Chapter 2

Survey of Related Research

Over the past few decades, a lot of research has been done in the field of declarative SQL query optimization. There are excellent surveys such as [9] and [20] which give a comprehensive view of the progress achieved over the years and an interested reader can refer the same. For a survey of work on query optimization related to *optimizer diagrams*, we refer the reader to [21]. In this chapter, we first give an overview of the basic concepts of query optimization. We then motivate our study of join orders by the importance of join order enumeration and provide details about the recent research advances. Further, we provide a short description of the available MDS techniques and their applicability to our semantic based coloring problem. Finally, we elaborate on the challenges in optimizing XML queries and briefly describe the XQuery optimizer we have worked with in our study.

2.1 Challenges of SQL Query Optimization

The job of the SQL query optimizer is to take parsed SQL as input and generate an efficient execution plan. It is computationally very challenging to produce an efficient plan for a given query, since the search space of possible plans can be of the order of $\Omega(3^n)$ [44], where n is the number of base relations in the given query.

There are some important considerations in the design of an optimizer, some of which are elaborated below.

2.1.1 Strategies for Plan Selection

There are different plan selection strategies including randomized algorithms and use of set of heuristic rules. One of the important strategies which is widely incorporated in commercial optimizers is the exhaustive enumeration of search space using a cost based approach. The seminal paper by Selinger et al. [42] (System-R project) was one of the key contributors in cost based query optimization. They considered only the search space of left deep trees, but the general idea of their algorithm can be used to explore the space of other tree spaces (right-deep, zig-zag and bushy) as well.

System-R's cost based optimizer uses dynamic programming to efficiently find a good plan. In this approach, optimal solutions to sub-problems are combined to form the optimal solution of the main problem. This is the principle of optimality [54]. The dynamic programming approach performs well for queries which do not have more than around a dozen base relations, after which it becomes infeasible due to the exponential increase in the search space complexity.

2.1.2 Join Orders

As discussed previously, dynamic programming techniques are extensively used to efficiently find a good plan. One of the most crucial tasks in this process is to find the optimal join order and significant attention has been given to studies on join order enumeration in the past.

Ono and Lohman [37] have elaborated on the computational complexity of enumerating join orders for a single select-project-join query block for a variety of join graphs. Further, more recently in 2006, Moerkotte and Neumann [34] showed that traditional bottom-up join enumeration algorithms are far worse than than the lower bound given by Ono and Lohman and went on to describe a new bottom-up enumeration algorithm that is optimal (conforms to the lower bound given by Ono and Lohman) over any join graph.

Thus, join order enumeration being very important, but computationally expensive, any savings during optimization would be highly beneficial. So, if the number of join orders in any plan diagram turns small, the join order templates can be fed as skeletons to the optimization process, thus saving a lot of optimization cost.

To our knowledge no prior work has been done on join order analysis with respect to plan

diagrams. However, the concept of plan reuse was used extensively in [17], even though not in the plan diagram setting. Another closely related work is by Satyanarayana et al. [48], where they re-use plans as in [17], but they characterize the plans in the form of join trees and join order templates, gather statistics of the join trees and join order templates, and use them to discover the optimal plan of a new query.

2.1.3 Refinements of Plan Choices at Run-time

Inaccurate estimates of various parameters, lead to poor plan choices by the cost based query optimizer. When moderately accurate values of some of these parameters are available at run time and if their values remain constant throughout the execution time, then certain strategies have been proposed. One of them is to find the best execution plan for all possible values of the parameters and lookup the best plan for the current parameter values at runtime. With the help of structure-based reduction, we can find the best join order for all possible values of the parameters, flesh out the join orders at runtime and find the best plan among them. If the join order cardinalities of plan diagrams turn out to be low, this can reduce much more optimization time as compared to costing hundreds of plans at run-time. So, structure-based reduction provides an intermediate strategy between fresh optimization of a query and costing all POSP (Parametric Optimal Set of Plans) plans.

2.2 Industrial Strength SQL Optimizers

In order to study the cost distribution of query plans in industrial strength SQL optimizers Waas and Galindo [49] have devised various algorithms. From the observations made using these algorithms, they found that only 1% of the plans in the entire search space of plans, have cost within twice the optimum cost. Also, they found that for queries having more than 4-5 relations, the distributions "*correspond to Gamma-distributions with shape parameter close to 1*". On a different note, Reddy and Haritsa [38] have studied the optimality space of industrial strength optimizers using the "plan diagram" notion. They observed that optimizers make extremely fine grained choices with changes in the query plan chosen, with minute changes in selectivity. They

also observed that optimal plans often have irregular boundaries and highly intricate patterns, indicating strong non-linear cost models. In the follow up of this work [10], Harish et al. investigated the plan reduction issue from the theoretical, statistical and empirical perspectives. They explored a cost-based approach to the reduction problem where given a plan diagram and a cost-increase-threshold (λ) specified by the user, the plan diagram was recolored to a simpler picture, featuring only a subset of the original plans, such that the cost of any recolored query point does not increase by more than λ percent, relative to its original cost. In our work, we take a different tack of reducing plan diagrams from a *structure-based* perspective. We coalesce all plans that have the same join order into a single entity, and so after the reduction, there are as many plans in the diagram as the number of unique join orders. If the join order cardinalities turn out to be low, then join order caching is an immediate application. Among the many benefits of structure-based reduction as opposed to the cost-based approach is that we always provide the optimal plan at every query location and that it is applicable for all types of queries, since we do not make any assumptions about the underlying cost model.

2.3 Multidimensional Scaling

Multidimensional Scaling (MDS) comprises a family of techniques and has been researched extensively. MDS has been broadly classified into *metric* MDS and *non-metric* MDS [46]. The basic distinction between the two is the level of their input data. If only the rank-order of the input dissimilarities is considered informative, then *non metric* MDS has to be considered and if the input dissimilarities are related to the output proximities by a specific continuous function, then *metric* MDS techniques need to be employed. But it is important to note that Non-Metric MDS techniques can work on higher level data as well.

Among the various techniques developed, one of the more popular techniques for Metric Multidimensional techniques is the SMACOF [7] (Scaling by majorizing a complicated function) algorithm which has guarantees on the rate of convergence. Also, one of the oldest and well known techniques for non-metric MDS is the Steepest Descent approach by Kruskal [29, 30]. There are various other techniques and many other features of MDS which

have been thoroughly investigated in [7].

MDS has been used extensively for various purposes, primarily for visualization of data, but from a database perspective, it is used for visualizing semantic differences in plan trees for the first time.

2.4 XML Query Processing

XML has become the de facto standard for information interchange. With this being the case, databases are now being largely used to store and query the XML documents. There are many approaches to storing XML data

1. Shredding the XML documents to be stored in relational columns. This implies that the user given XML query will have to be converted to its equivalent SQL and also the results returned will be converted to their equivalent XML format if needed.

2. Store the XML as BLOB (Binary large object) or CLOB (Character large object) data types in relational columns. With this approach, it is generally not feasible to have indexes and very often, all documents may need to be processed in their entirety with either XQuery or SQL, whichever the engine supports.

3. Develop specialized data management for XML, also known as native XML databases.

4. A hybrid approach where XML is stored natively along with relational data.

Regarding the querying of XML documents in databases, the desired characteristics of an XML query language were outlined in [63]. The development of some XML query languages are highly influenced by his criteria. Some of the important issues addressed in his proposal include XML output of the query, independence of schema, schema exploitation if possible, and optimized query operations.

XQuery [65] has been established as the W3C recommendation in 2007 for querying XML. However, prior to the establishment of the W3C standard, there had been several studies proposing query languages for XML, including XML-QL [13], Lorel [1], Quilt [8] and XQL [60].

As given in [45], XQuery semantics are highly influenced by Quilt and XPath [64]. XQuery uses XPath for path expressions and FLWOR structure for describing the whole query. Also, it embodies many of the desired characteristics elaborated in [63].

There are several challenges in querying XML data. They include XML data being heterogeneous and hierarchical in nature as opposed to homogeneous relational data which has flat schemas, NULL values in XML being indicated by the absence of certain elements, with no need to be specified explicitly and the varying of XML schemas from document to document, and sometimes within the same column. Also, XQuery, and in general XML querying languages are founded on monadic second order logic, as opposed to the first order logic of relational DBMS and hence have significantly more expressive power than their relational counterparts. This adds to the challenges of optimizing XML queries. The presence of such complexities enforces the need for the analysis of XML query optimizers, which motivates our work to extend the plan diagram notion to the XML world.

The first cost-based XML optimizer was designed and implemented for the Lore system [31, 32]. The query language used in this system is Lorel which is an extension of OQL. They developed new indexing schemes and database statistics which describe the shape of the data graph. They also described plan enumeration strategies with numerous access methods using logical and physical query plans and a cost model.

Following this, a lot of research has been done in XQuery optimization with the development of several XML compliant database systems. [19], [22], [18], [45] and [16] provide surveys on XML query languages and XML query processing. Also, an overview of XQuery optimization techniques and its implementation is given in [5]. We now briefly describe the optimization techniques and implementation details of the XQuery processing system in DB_XML, which is our optimizer of interest. DB_XML has Native XML support along with relational data and so now supports XQuery along with SQL. This hybrid characteristic naturally brings support for SQL/XML language. They even have a hybrid optimizer where XQuery and SQL optimization is done in a unified framework. Given this, XQuery optimization largely reuses the already in-place relational optimization techniques. They use holistic algorithms and heuristics to process complex path expression within a single operator. These algorithms were designed to reduce

the search space without compromising on the quality of plans. They also use structural and value indexes in a combined fashion during plan generation. To this effect, they have introduced new operators such as XSCAN, XANDOR and XISCAN. An excellent in-detail description of the cost based optimization done in DB_XML is given in [4].

Chapter 3

Structure-based Reduction of Plan Diagrams

As highlighted in Chapter 1, Section 1.2, plan diagrams often feature non-convex and complicated patterns along with high plan cardinalities. One of the approaches to simplify the plan diagram is through a cost-based process, which was investigated in [10], where plans were swallowed by sibling plans without increasing the cost of any individual query point in the plan diagram by more than a user-specified threshold. In this chapter, we approach the plan simplification process through an alternative structure-based procedure, where all plans that have the same *join order* are merged into a single entity. We adopt this merging procedure since determining the join order is one of the most crucial tasks of optimization and join order caching is a potential application, which provides the "best" plan along with savings in optimization time when the join order cardinality is low or moderate. We first elaborate on the structure-based reduction process. Then, we discuss the benefits of structure-based reduction relative to cost-based reduction along with the applications of structure-based reduction. Further, we present experimental results obtained by applying structure-based reduction on a suite of plan diagrams. While the results indicate that the resulting join order cardinalities are low or moderate in many cases, we also encounter situations wherein the cardinality is high. For the latter case, we present the SRE (Small Relation Elimination) heuristic, that removes small sized relations in the join orders to bring down the join order cardinality. Even though the plan optimality

is no longer guaranteed, we present experimental results, which show that the cost-increase of any query point is within acceptable limits. Finally, we study the occurrence of common join orders in the plan diagrams produced by different commercial engines.

3.1 Join Orders

We now provide the formal definition of a join order for a query execution plan.

DEFINITION 1 *Join Order.*

The join order of a query execution plan is the order in which the relations of the query are joined, with each join being annotated by the join-type and the join predicate(s) applied at that join.

The *join-type* can be any of the various joins possible, including *Natural Join*, *Left Outer Join*, *Right Outer Join* and *Full Outer Join*.

Given a query execution plan tree, its join order is determined by an **inorder** traversal of the tree. Consider, for example, the plan tree shown in Figure 3.1. The annotated join-order for this plan obtained after its inorder traversal would be:

$(\text{CUSTOMER} \bowtie_{(c\_custkey=o\_custkey)} \text{ORDERS})) \bowtie_{(l\_orderkey=o\_orderkey)} \text{LINEITEM}$

Here, CUSTOMER and ORDERS relations are first joined together using the join predicate *c_custkey* = *o_custkey*, the result of which is joined with the relation LINEITEM using *l_orderkey* = *o_orderkey* as the join predicate. Both the joins are Natural joins as indicated by the $\bowtie$ symbol.

3.2 Overview of Structure-based Reduction

It is well known that determining the optimal join order is the most computationally challenging task during the query optimization process and that this decision has a very profound effect on the quality of the plan chosen. So, in the structure-based reduction process, we merge all plans with the same annotated join order into a single entity. By same join order, we mean that the annotated join orders of the plans are the same. Thus, after the reduction process, there would be as many colors in the plan diagram as the number of unique join-orders. If the resulting diagram turns out to be anorexic, then structure-based reduction is a potent alternative

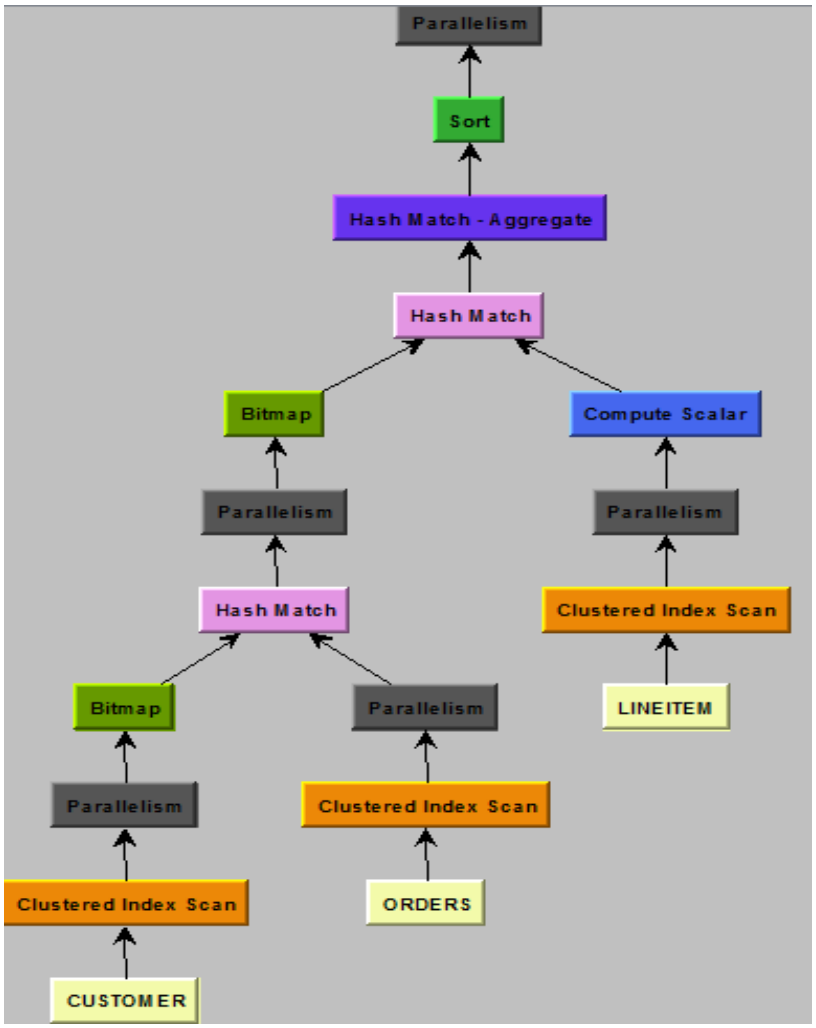
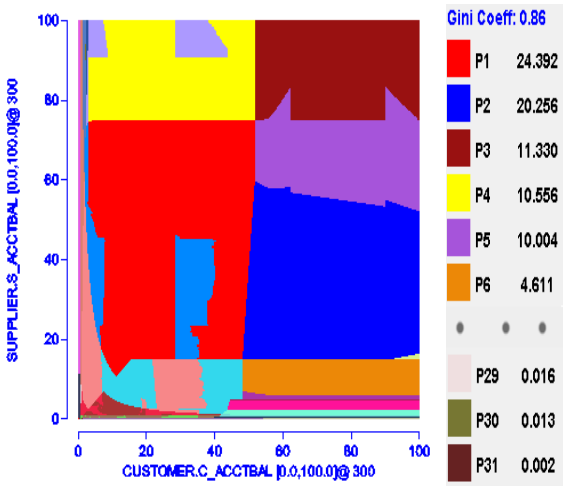


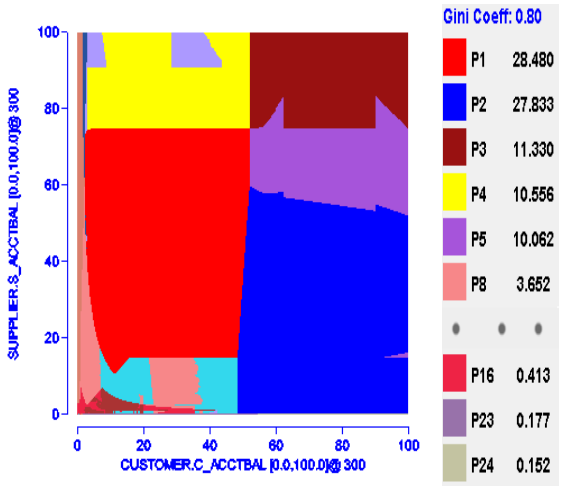
Figure 3.1: Example Plan Tree for Join Order

to cost-based reduction, but without the cost increase overheads that are inherent in cost-based reduction techniques.

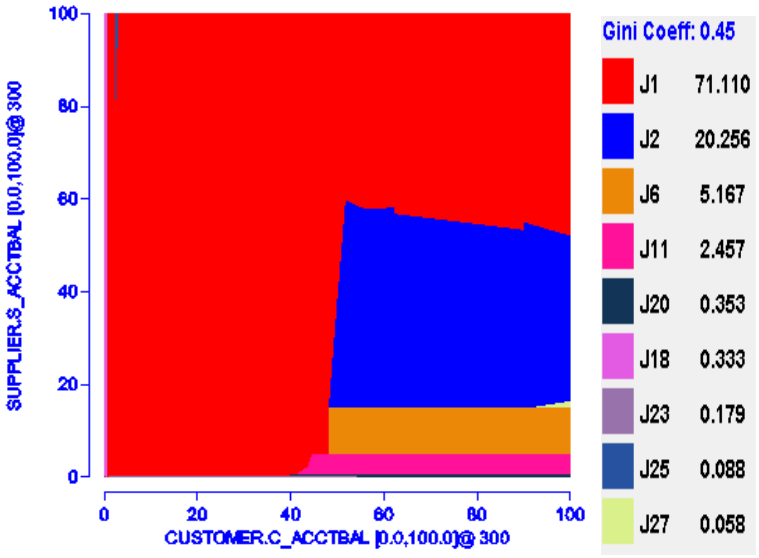
As an example, consider the plan diagram of Figure 3.2(a) consisting of 31 plans. A cost-based reduction of this plan diagram at a cost increase threshold of 20% is shown in Figure 3.2(b). The number of plans has reduced to 14, but most of the geometries of the plans in the plan diagram remain intact. But the structure-based reduction of the same plan diagram, which is shown in Figure 3.2(c), brings down the number of plan skeletons to 9, and additionally, the diagram is much "cleaner", with the complicated plan boundaries being now removed.



(a) **Plan Diagram**



(b) **Cost-based Reduction** ($\lambda = 20\%$)



(c) **Structure-based Reduction**

Figure 3.2: Example for Structure-based Reduction

Some of the join orders found in this plan diagram are listed below. Their tree representations are shown in Figure 3.3.

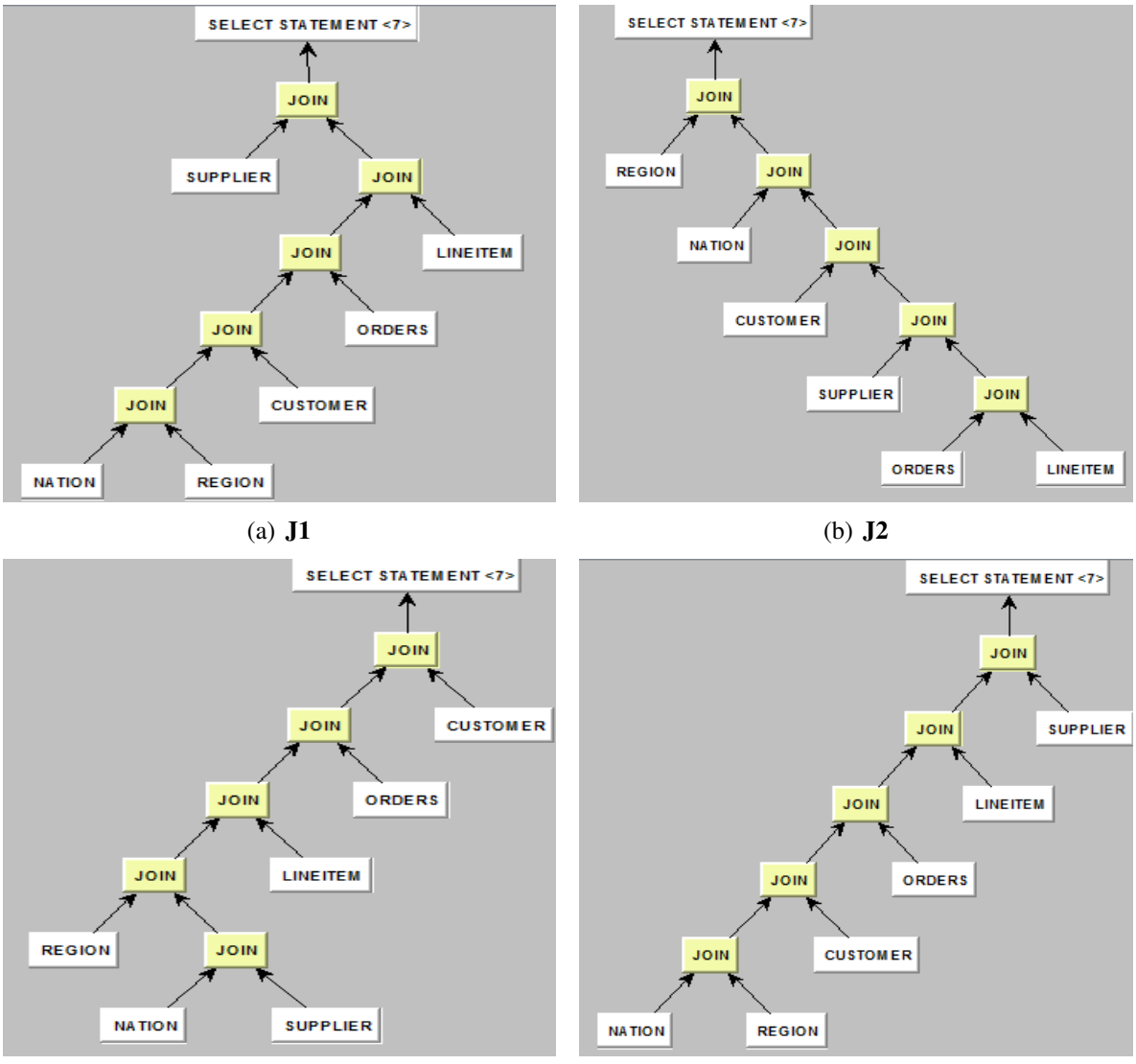
J1 - (SUPPLIER $\bowtie_{(s\_nationkey=n\_nationkey,s\_suppkey=l\_suppkey)}$ ((((NATION $\bowtie_{(n\_regionkey=r\_regionkey)}$ REGION) $\bowtie_{(n\_nationkey=c\_nationkey)}$ CUSTOMER) $\bowtie_{(c\_custkey=o\_custkey)}$ ORDERS) $\bowtie_{(l\_orderkey=o\_orderkey)}$ LINEITEM))

J2 - (REGION $\bowtie_{(n\_regionkey=r\_regionkey)}$ (NATION $\bowtie_{(s\_nationkey=n\_nationkey)}$ (CUSTOMER $\bowtie_{(c\_custkey=o\_custkey,c\_nationkey=s\_nationkey)}$ (SUPPLIER $\bowtie_{(s\_suppkey=l\_suppkey)}$ (ORDERS $\bowtie_{(l\_orderkey=o\_orderkey)}$ LINEITEM)))))

J11 - ((((REGION $\bowtie_{(n\_regionkey=r\_regionkey)}$ (NATION $\bowtie_{(s\_nationkey=n\_nationkey)}$ SUPPLIER)) $\bowtie_{(s\_suppkey=l\_suppkey)}$ LINEITEM) $\bowtie_{(l\_orderkey=o\_orderkey)}$ ORDERS) $\bowtie_{(c\_custkey=o\_custkey,c\_nationkey=n\_nationkey)}$ CUSTOMER)

J25 - (((((NATION $\bowtie_{(n\_regionkey=r\_regionkey)}$ REGION) $\bowtie_{(n\_nationkey=c\_nationkey)}$ CUSTOMER) $\bowtie_{(c\_custkey=o\_custkey)}$ ORDERS) $\bowtie_{(l\_orderkey=o\_orderkey)}$ LINEITEM) $\bowtie_{(s\_nationkey=n\_nationkey,s\_suppkey=l\_suppkey)}$ SUPPLIER)

Even though the join order cardinality of the plan diagram is just 9, it covers a spectrum of disjoint join orders, consisting of left-deep (Figure 3.3(d) corresponding to J25), zig-zag, where each join has at least one input to be a base relation, and the tree is neither left-deep nor right-deep (J1 and J11 corresponding to Figures 3.3(a) and 3.3(c)) and right-deep trees (J2 shown in Figure 3.3(b)). While no bushy trees were found in this particular case, they were found in many other plan diagrams, some of which are discussed later. Also, all three tree-types that are exhibited in the plan diagram are also present in the cost-based reduced diagram that has only 14 plans.



(a) **J1** (b) **J2**

(c) **J11** (d) **J25**

Figure 3.3: Example Join Orders

3.3 Benefits and Applications of Structure-based Reduction

We now discuss the benefits of structure-based reduction as compared to cost-based reduction, which makes it very useful.

- No assumptions about the underlying cost model are made, thus making it suitable for **all** kinds of query templates. Cost-based reduction, on the other hand *requires* plan cost monotonicity to hold.
- Works solely through the standard "hint" mechanism supported by most engines. Some cost-based reduction schemes require advanced features like foreign plan costing (FPC) for their functioning.
- If it is known that the database instance and the optimizer/engine have not changed since the time of plan diagram generation, structure-based reduction when used for join order caching, modulo the SRE heuristic (Section 3.4.1), provides the plan as would have been chosen by the optimizer ("optimal plan"). However, cost-based reduction procedures provide plans that are within a cost-increase threshold of the optimal plan. On the other hand, if either the database instance or the engine has changed, structure-based reduction provides consistent join orders (i.e., as was chosen previously by the optimizer), which is sometimes an important requirement in many industrial settings. Cost-based reduction procedures, however, could suggest a new plan.
- When values of certain parameters cannot be predicted at compile time, but can be known at run-time, join order caching can be used as part of a run-time plan refinement strategy to provide better plan choices along with savings in optimization time.

Applications. As an immediate application of the structure-based reduction scheme, we discuss join-order caching. Here, when a new query corresponding to a location in the plan diagram is issued, we do not carry out a fresh optimization of the query. Instead, the join-orders are used as "skeletons" that are fed to the optimization process to be fleshed out into complete plans, which are in turn costed to find the best plan. This process of feeding of just the join-orders to the optimization process is achieved through the "hinting" mechanism provided by most database

engines. Among the functionalities provided by the hinting mechanism, users are typically allowed to specify access methods (such as *Index Scan* and *Table Scan*), join orders and join operators (such as *Hash-Join*, *Merge-Join* and *NestedLoop-Join*). For our purposes we need to specify only the join orders.

We have empirically found that fresh optimization of a query requires about a factor of 20 more time than fleshing out a complete plan from a given join-order. This observation implies that join order caching is certainly beneficial for plan diagrams whose join-order cardinality is low – around 10. For the plan diagram in Figure 3.2(a) that has 31 plans, the structure-based reduced diagram, which is shown in Figure 3.2(c) has only 9 join orders. Join order caching would be highly beneficial in this case. Further, the time required for fleshing out the skeletons includes overheads such as unnecessary query re-writing and hint validation, which if eliminated, *by incorporating join-order caching into the optimizer*, would reduce the time taken even further. This makes join-order caching viable even for plan diagrams with moderately high cardinalities - around 40. Finally, when faced with very high join-order cardinalities, we propose a heuristic-based solution – SRE (Small Relation Elimination) – to reduce it to anorexic levels, which is outlined later in the chapter.

One more interesting use case is when we fail to get anorexic diagrams using cost based reduction schemes. Structural reduction can be applied on top of the cost based reduction, giving us the desired results. The plan diagram of Figure 3.2(a), after cost-based reduction yields 14 plans as shown in Figure 3.2(b) at $\lambda = 20\%$. Application of structure-based reduction on top of this leaves us with just 5 join orders as shown in Figure 3.4 with the bonus of obtaining a "clean" diagram.

Also, extrapolating from the above idea, this two-level reduction scheme may prove to be helpful in cases where we prefer λ to be less than 20%, generally required to get anorexia in cost-based reduction techniques. In such scenarios, this two level application of reduction might get us a low number of plans along with lesser cost-increase thresholds. As an example, consider again the plan diagram of Figure 3.2(a). If we desire a λ of 5%, the number of plans remaining after cost-based reduction is 23. After applying structure-based reduction on this result, we get 7 join orders and the diagram looks very similar to Figure 3.2(c). Considering plan costing takes

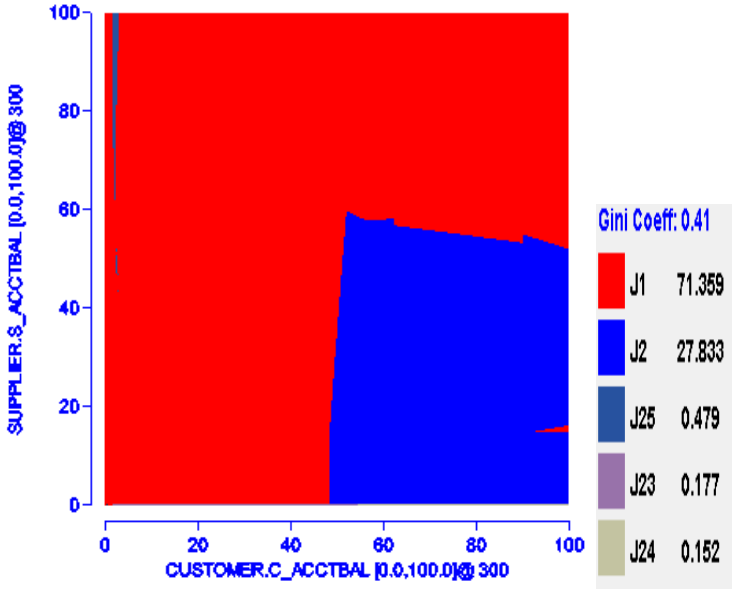


Figure 3.4: Structure-based Reduction of a Cost-based Reduced diagram

far less time than fleshing out skeletons, this strategy can be very useful when the ratio of join orders remaining after structure-based reduction of the cost reduced diagram to the number of plans remaining after only cost-based reduction is less than the ratio of time required for plan costing to the time required fleshing out a join order. If join order caching is internalized in the optimizers, as discussed earlier, the second ratio would increase, thus providing more room for the strategy to be helpful.

3.4 Experimental Evaluation of Structure-based Reduction

In this section, we present results obtained by applying structure-based reduction on plan diagrams that are produced on two industrial-strength commercial optimizers. These optimizers are anonymously referred to as Opt A and Opt B [1], with Opt A being a top-down optimizer and Opt B, a bottom-up one.

Our experimental setup consists of running the database engines and obtaining plan diagrams on an Intel Core 2 Duo machine having a clock speed of 2.10 GHz (T8100), with 4GB

[1]Note for review purposes only: The optimizers are Microsoft SQL Server 2008 and DB2 9.7.

QT	No. of plans	No. of join orders	No. of pruned join orders
2	80	28	20
8	128	52	16
9	114	32	21

Table 3.1: Structure-based Reduction Statistics for Opt A

QT	No. of plans	No. of join orders	No. of pruned join orders
2	15	10	5
8	40	29	11
9	67	54	34

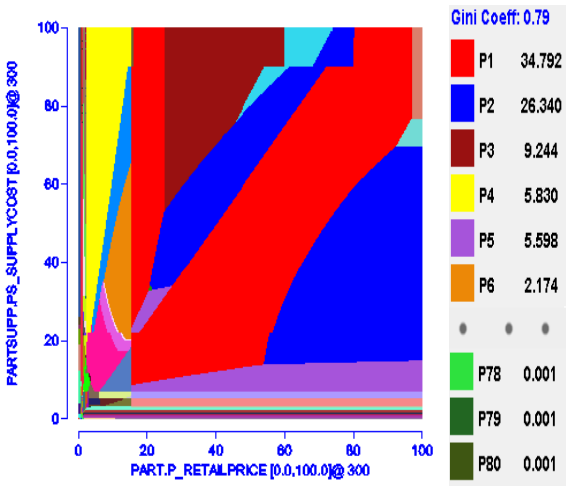
Table 3.2: Structure-based Reduction Statistics for Opt B

RAM (3GB usable), running a copy of Windows 7 Professional (32 bit).

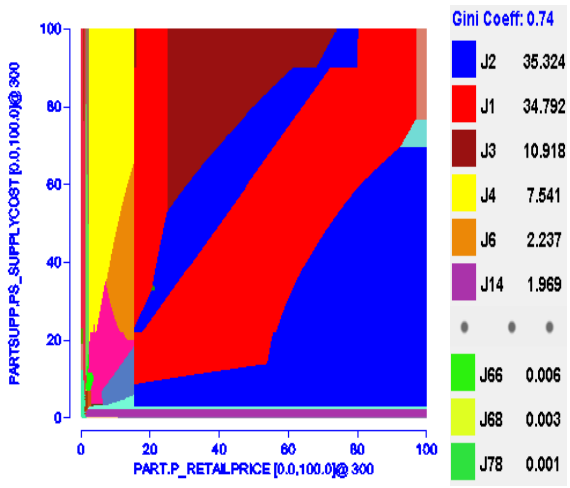
Our results and analysis are based on our experiments using parameterized versions of three queries from the TPC-H benchmark [62]. These three queries - 2, 8 and 9 of the TPC-H benchmark, are referred to as Query Templates (QT) 2, 8 and 9, respectively and are provided in Figures A.1, A.3 and A.4, respectively. The templates for these queries were chosen as they yield a fairly large number of plans as compared to the rest of the queries in both the engines. Also, the templates are very complex as they involve a large number of tables and exhibit nesting in the *SELECT*, *FROM* and *WHERE* clauses.

In Tables 3.1 and 3.2, we have enumerated the number of unique join-orders in QT2, QT8 and QT9 for Opt A and Opt B, respectively. We see that there is a substantial decrease in the number of entities in the plan diagram in most cases. The plan diagrams and the respective structure-based reduced diagrams corresponding to the three templates are shown in Figures 3.5 and 3.6 for Opt A and Opt B, respectively.

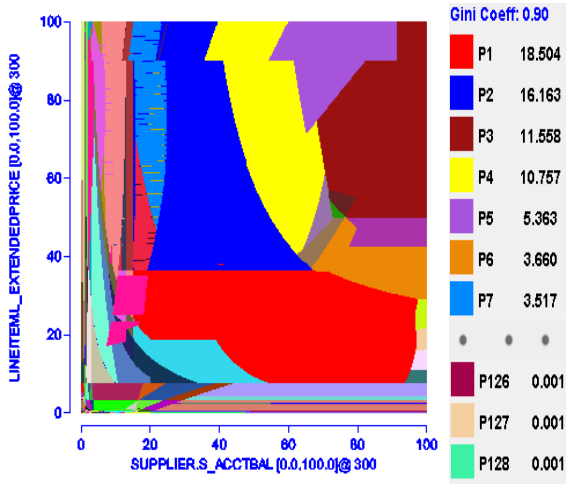
One characteristic feature in all the cases is that the structure-based reduced diagram looks "cleaner" than the plan diagram. This is quite evident in the structure-based reduced diagram of QT8 - Opt A, shown in Figure 3.5(d), where the complicated spatial layouts of the plan optimality regions near the axes are replaced with relatively smoother boundaries.



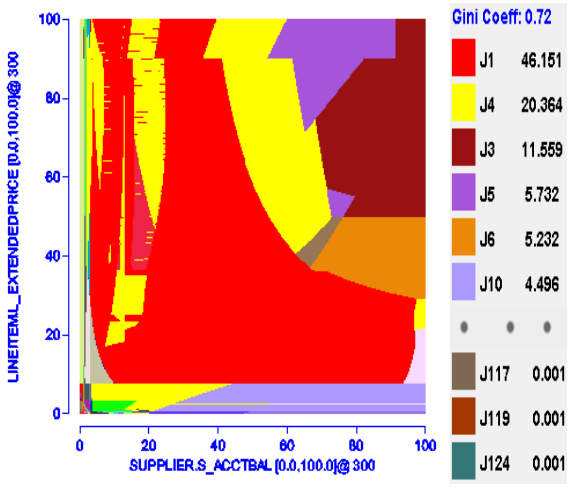
(a) **Plan Diagram for QT2**



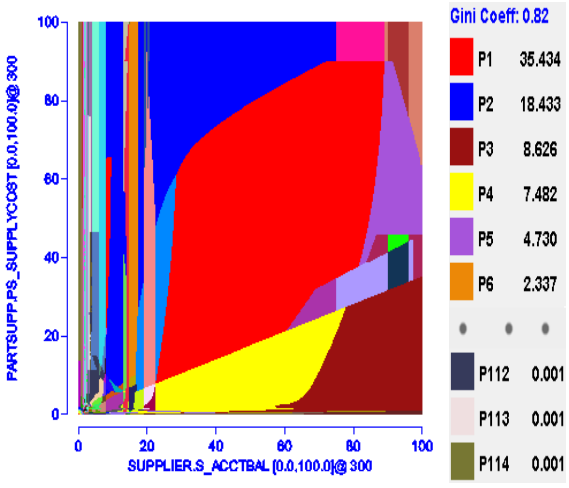
(b) **Structure-based Reduced Diagram for QT2**



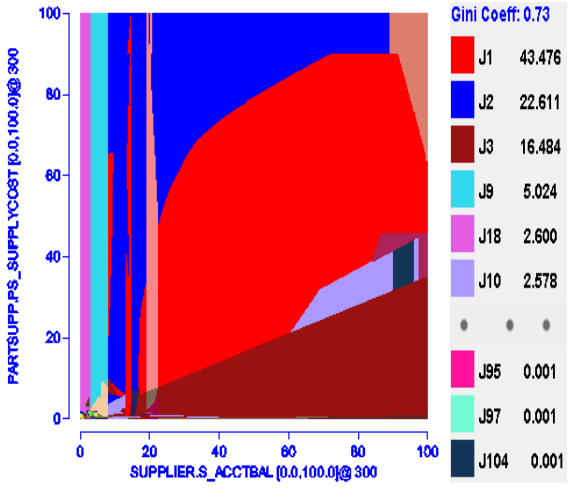
(c) **Plan Diagram for QT8**



(d) **Structure-based Reduced Diagram for QT8**

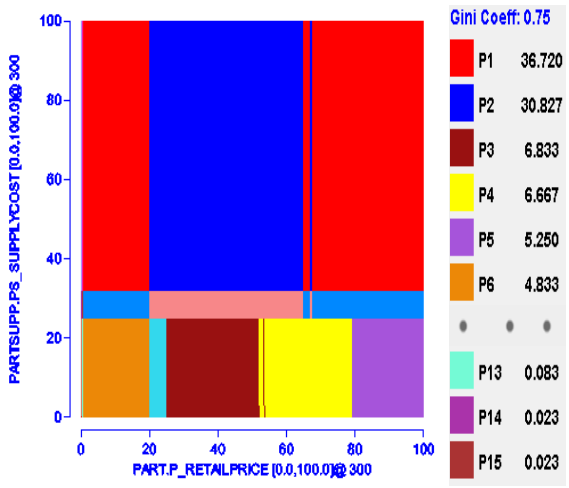


(e) **Plan Diagram for QT9**

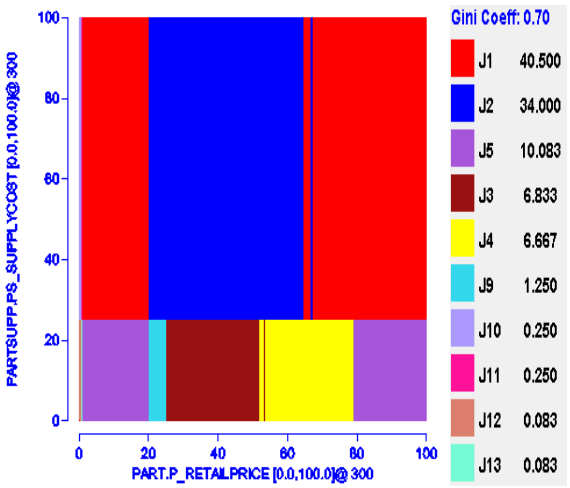


(f) **Structure-based Reduced Diagram for QT9**

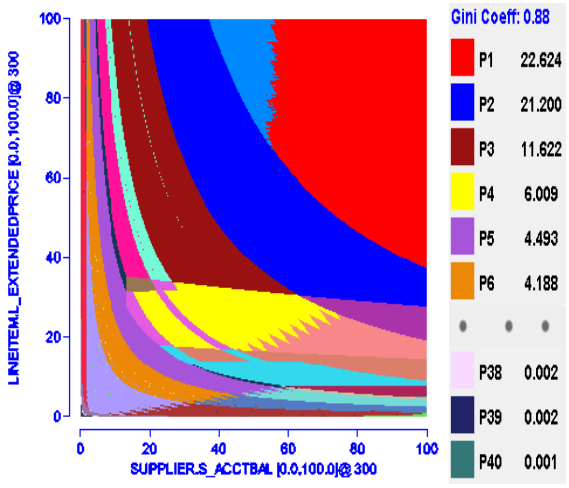
Figure 3.5: Structure-based Reduction for Opt A



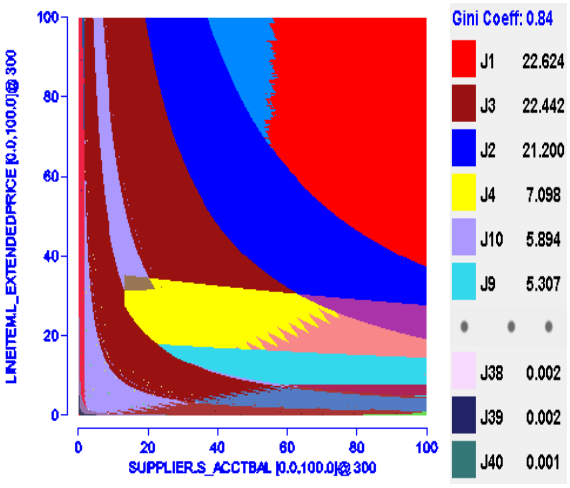
(a) **Plan Diagram for QT2**



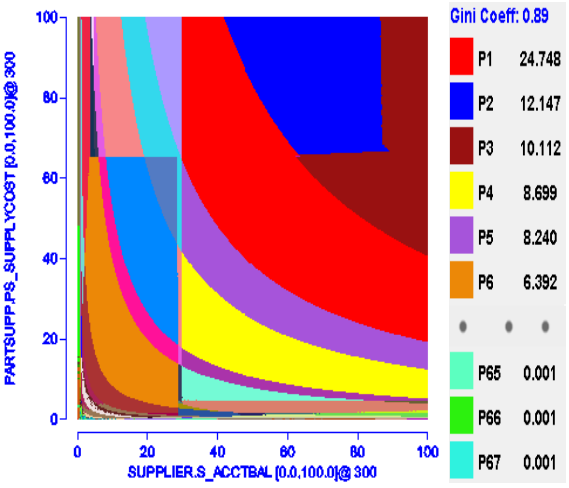
(b) **Structure-based Reduced Diagram for QT2**



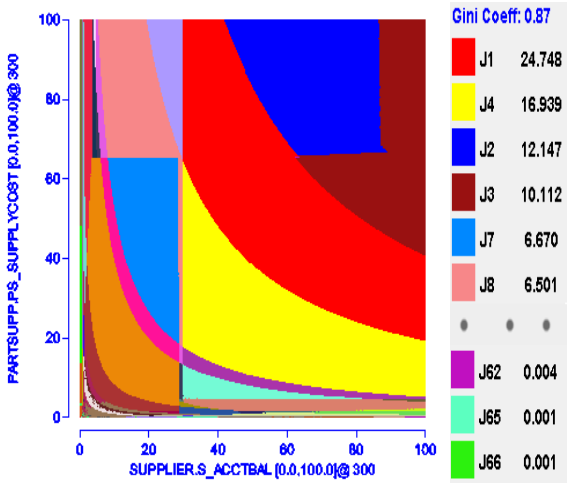
(c) **Plan Diagram for QT8**



(d) **Structure-based Reduced Diagram for QT8**



(e) **Plan Diagram for QT9**



(f) **Structure-based Reduced Diagram for QT9**

Figure 3.6: Structure-based Reduction for Opt B

QT	left-deep	right-deep	bushy	zig-zag
2	1	0	15	12
8	7	0	14	31
9	1	0	9	22

Table 3.3: Tree-types found in Opt A

QT	left-deep	right-deep	bushy	zig-zag
2	0	0	10	0
8	1	0	11	17
9	0	0	15	39

Table 3.4: Tree-types found in Opt B

After drilling down to the plan structures, we tabulate in Tables 3.3 and 3.4, the tree-types found in the all three QTs for Opt A and Opt B, respectively. We notice that there are no right-deep join orders present in any of the three templates for both the optimizers. The one left-deep join order of QT2 - Opt A, corresponding to J14, occupies around 2% of the diagram space and is found to be the purple horizontal patch near the X-axis in Figure 3.5(b), while those of QT9 - Opt A (J86) and QT8 - Opt B (J28), shown in Figures 3.5(f) and 3.6(d), respectively, occupy less than 0.05% of the space and are not visible to the naked eye.

All experimental results presented here are on two dimensional query templates, but the techniques remain applicable to higher dimensional templates too.

3.4.1 Heuristic-based Solution

We discussed previously that structure-based reduction has useful applications to join order caching. However, when the join order cardinalities of plan diagrams turn out to be very high, it is not beneficial to directly use the structure-based reduction procedure. To overcome this drawback, we discuss a heuristic-based solution – SRE (Small Relation Elimination) – to structure-based reduction that can be applied when the join order cardinality is substantially high, in order to make join order caching feasible.

Analysis on the join-orders present in the plan diagrams showed that, in cases where the

number of join-orders were high, there existed many join orders which differed only in the join of *smaller* relations. So, based on the hypothesis that large number of join orders are because of the variations in the joins of *smaller* relations and that the change in the join order with respect to them does not have a huge impact on the join order quality, we propose the following heuristic-based solution as a recourse when high join-order cardinality is encountered. From the join orders initially obtained from the plans, we remove the relatively smaller sized relations, thereby obtaining a pruned set of partial join-orders. Then, for each unique partial join order, we assign a representative complete join order, chosen randomly from among the complete join orders that are built from the partial join orders. As an example, let us look the plan diagram of QT8 - Opt A, shown in Figure 3.5(c), which has 52 join orders. Consider the set of join orders shown below – J27, J69, J116 – that are taken from the plan diagram. Their join predicates are dropped for ease of presentation and their tree-representations are shown in Figures 3.7(a), 3.7(b) and 3.7(c), respectively.

J27 - (PART ⋈ ((SUPPLIER ⋈ NATION) ⋈ ((((LINEITEM ⋈ ORDERS) ⋈ CUSTOMER) ⋈ NATION) ⋈ REGION)))

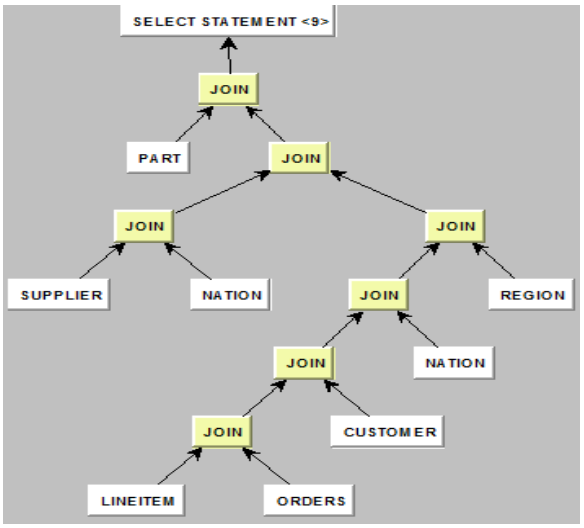
J69 - (PART ⋈ ((SUPPLIER ⋈ NATION) ⋈ (REGION ⋈ (NATION ⋈ ((LINEITEM ⋈ ORDERS) ⋈ CUSTOMER)))))

J116 - (PART ⋈ ((SUPPLIER ⋈ NATION) ⋈ (REGION ⋈ (((LINEITEM ⋈ ORDERS) ⋈ CUSTOMER) ⋈ NATION))))

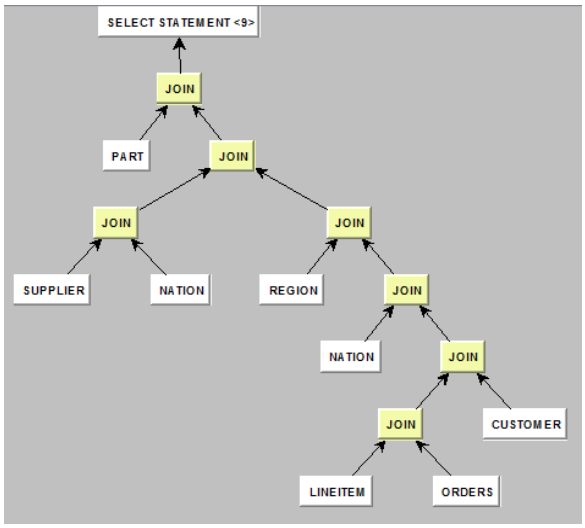
After removing the relations NATION and REGION from all three join orders we get -

Pruned Join Order - (PARTS ⋈ (SUPPLIER ⋈ ((LINEITEM ⋈ ORDERS) ⋈ CUSTOMER)))

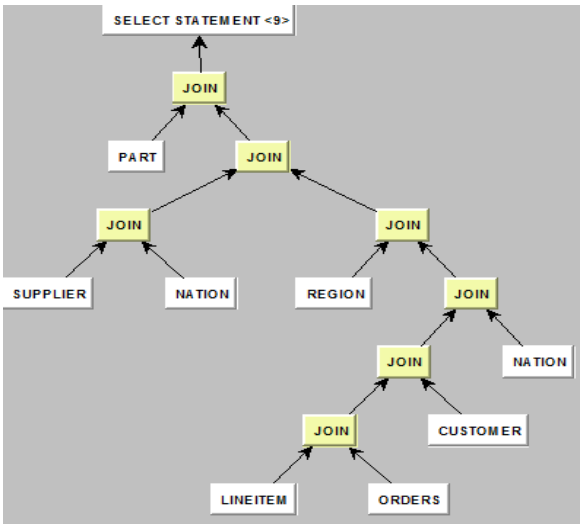
This is the partial join order to which any of the three complete join orders can be assigned as the representative join order. Figure 3.7(d) shows the structure of the pruned join order.



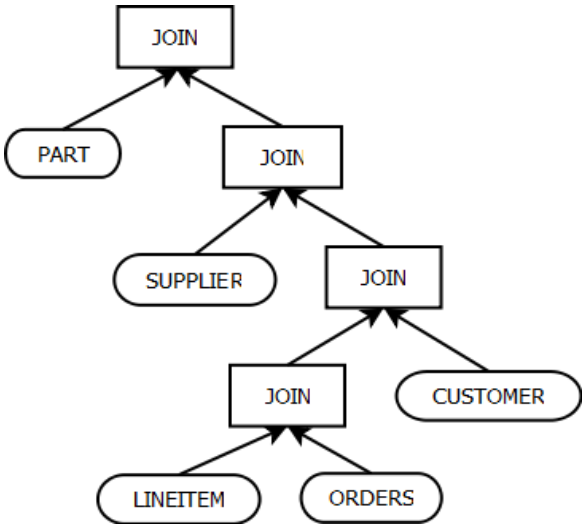
(a) **J27**



(b) **J69**



(c) **J116**



(d) **Pruned Join Order**

Figure 3.7: Example Join Orders of QT8 - Opt A

Now, when a query located in the plan diagram has to be optimized, we evaluate the representative join orders corresponding to each unique partial join order and choose the one yielding the least cost.

Experimental evaluation of this heuristic using the experimental setup described previously did yield favourable results. The number of pruned join orders remaining in the plan diagrams of QT2, QT8 and QT9, after removing the smaller relations, are shown in the fourth columns of Tables 3.1 and 3.2 for Opt A and Opt B, respectively. It can be seen that the number of pruned join orders has come down to around 20 in most cases. It is important to note that, even after removal of the smaller relations, there are around 4-5 relations left in each of the queries and hence we still have hundreds of potential join orders in our search space. Also, it was sufficient to remove only 2 relations in most cases to get reduced join order cardinalities. Finally, the heuristic was evaluated by applying it to every query location in the plan diagram. We observed that the per-query cost increase was less than 30% for over 90% of the query locations in the plan diagram. Even for the remaining the 10% of query locations, the cost-increase was less than 100%, which is generally considered to be the highest acceptable cost-increase threshold. Currently, the evaluation of the heuristic was carried out only for QT8 and QT9 with Opt A. QT2 was not considered because it involves nesting in the *Where* clause of the query template and there is no readily available mechanism in Opt A to force join orders for such queries.

Also, join-order reduction followed by SRE and cost-based reduction followed by join-order reduction might seem comparable at first sight, but are two very different ways in which plan diagrams can be reduced. In particular,:

1. The time taken by cost-based reduction is directly proportional to the number of query points in the plan diagram, whereas for structure-based reduction it is proportional to the number of plans in the plan diagram. This makes SRE computationally less expensive than cost-based reduction followed by join order reduction.

2. Cost-based reduction strategies require the user to provide a cost increase threshold λ, and the recommended λ in order to obtain anorexic number of plans is around 20%. We have experimentally verified that the effective cost increase percentage through SRE is around 20%-30%. Thus, SRE is not materially worse than cost-based reduction in

QT	$A \cap B$	$A\_pruned \cap B\_pruned$
2	0	1
8	0	0
9	0	0

Table 3.5: Inter Engine Join Order Statistics

practice. However, SRE provides no guarantee on the maximum cost-increase of any point.

3. Also, if database engines provide mechanisms to "hint" only the partial join orders that are gotten through SRE, then it is possible to get the optimizer's optimal choice, while there are cost increases that are always associated with the cost-based reduction algorithms for the replaced set of points.

Equipped with the above results, we conclude that structure-based reduction scheme can be fruitfully employed for join-order caching, either directly, or through the application of the SRE heuristic.

3.5 Inter Engine Join Orders

In Section 3.4, we looked at the structure-based reduction process by enumerating the unique join orders found in individual plan diagrams. We now move on to exploring the common join orders between the two optimizers, Opt A and Opt B for the same three templates, QT2, QT8 and QT9.

In Table 3.5 we have listed down the inter engine join order statistics for QT2, QT8 and QT9. The second column corresponds to number of common complete join orders, and the third corresponds to common pruned join orders (join orders with smaller relations removed). The results as we can see, are quite surprising with *no intersection* at all in most cases. Even among the set of pruned join orders, with smaller relations being removed, the intersection still remains 0 for QT8 and QT9 and is very sparse (1) for QT2.

This disjoint behavior can be because of various reasons including different cost models,

variations in the implementations of different join algorithms and difference in the methods of storing and using statistics. Further investigations of this puzzling behavior comprises interesting future work.

Chapter 4

Semantic Coloring of Plan Diagrams

With the ever increasing complexity of SQL queries, there is an increase in the plan cardinalities and intricacies of plan boundaries in plan diagrams, the details of which were discussed in Chapter 1. So, analysis of the plan diagrams is certainly not easy. In the previous chapter, through structure-based, we broadly looked into the differences found between plans. Structure-based reduction provides a broad overview on the kinds of plans present, but in case we require to analyze structural differences between many plans, we still need to go through the cumbersome process of comparing plans manually. In this chapter, we alleviate this problem, by *semantically* coloring the plan diagrams. Specifically, we color the plan diagrams such that the differences in colors between any pair of plans, reflects the differences in their structure.

We begin with formally defining the semantic plan diagram coloring problem and then go on to describe the plan tree differencing metric adopted from [14, 52] to capture the plan tree differences quantitatively. We then develop a technique to achieve semantic coloring of plan diagrams. Specifically, we adapt Kruskal's *Iterative Steepest Descent* method, which is a multidimensional scaling (MDS) technique. Finally, we discuss the experimental results obtained by employing the adapted technique to color a suite of plan diagrams. We also provide analysis on the quality of coloring obtained by the proposed technique and the kind of plans present in the plan diagrams.

4.1 Problem Definition

The formal definition of the problem, which was provided in Chapter 1, is reiterated here.

Problem Statement: Consider an undirected complete graph $G(V, E)$ with m vertices, where each vertex $v_i \in V$ represents a POSP (Parametric Optimal Set of Plans) plan. A vertex v_i has an associated weight w_{v_i}, representing the fractional volume covered by the plan in the plan diagram. Each edge $e_{ij} \in E$ has an associated weight p_{ij} representing the structural distance between the plans at its vertices v_i and v_j. Also, each vertex pair (v_i, v_j) is assigned a weight w_{ij}. The objective now is to come up with an efficient algorithm that assigns a unique color to all the vertices in V such that, given any pair of vertices (v_i, v_j), their color distance c_{ij} is as close as possible to the weight p_{ij} of the edge joining them. Further, if a choice has to be made, vertex pairs that are assigned higher weights (w_{ij}) receive preferential treatment in obtaining a more accurate coloring for their distances. That is, our aim is to minimize (i.e. as close to 0 as possible) the following objective function:

$$\sum_{v_i=1}^{m-1} \sum_{v_j=v_i+1}^{m} w_{ij}(p_{ij} - c_{ij})^2$$

A variety of assignments are possible for the weights w_{ij}: For example, they can be $w_{v_i} + w_{v_j}$, giving preference to plan pairs with larger total area; or 1, which treats all vertex pairs equally; or, $\frac{1}{w_{v_i} + w_{v_j}}$, where small sized plans are preferentially colored better.

4.2 Plan Distance Metrics

To calculate plan tree differences, we employed the plan tree differencing strategy from [14, 52], the details of which are revisited below.

Let the operator trees corresponding to a pair of plans p_i and p_j be denoted by T_i and T_j, respectively. Our comparison strategy is based on identifying and mapping similar operator nodes in the two trees. In the following description, the term *branch* is used to refer to any

directed chain of unary-input nodes between leaf and a binary node, or between a pair of binary-input nodes, in these trees. Branches are directed from the lower node to the higher node, modelling the direction of data propagation. The matching proceeds as follows:

1. First, all the leaf nodes (relations) and all the binary-input nodes (typically join nodes) are identified for T_i and T_j.

2. A leaf of T_i is matched with a leaf of T_j if and only if they both have the same relation name. In the situation that there are multiple matches available (that is, if the same relation name appears in multiple leaves), an edit-distance computation is made between the branches of all pairs of matching leaves between T_i and T_j. The assignments are then made in increasing order of edit-distances.

3. A binary node of T_i is matched with a binary node of T_j if the set of base relations that are processed is the same. If the node operator names and the left and right inputs are identical (in terms of base relations), the nodes are made white. However, if the node operator names are different, or if the left and right input relation subsets are different, then the nodes are colored.

4. A minimal edit-distance computation is made between the branches arising out of each pair of matched nodes, and the nodes that have to be added or deleted, if any, in order to make the branches identical, are colored. Unmodified nodes, on the other hand, are matched with their counterparts in the sibling tree and made white.

5. Finally, each pair of matched nodes is assigned the same unique number in both trees. For example, the number 15 is assigned to the final join node, representing the composite relation formed by the join of all the base relations, in each tree.

Plan Differencing Metric. We now describe the procedure to quantify plan-tree differences. Our formulation uses $|T_i|$ and $|T_j|$ to represent the number of nodes in plan-trees T_i and T_j, respectively, and $|T_i \cap T_j|$ to denote the number of matching nodes between the trees. The white nodes depict matching nodes, whereas the colored nodes represent distinct nodes. For

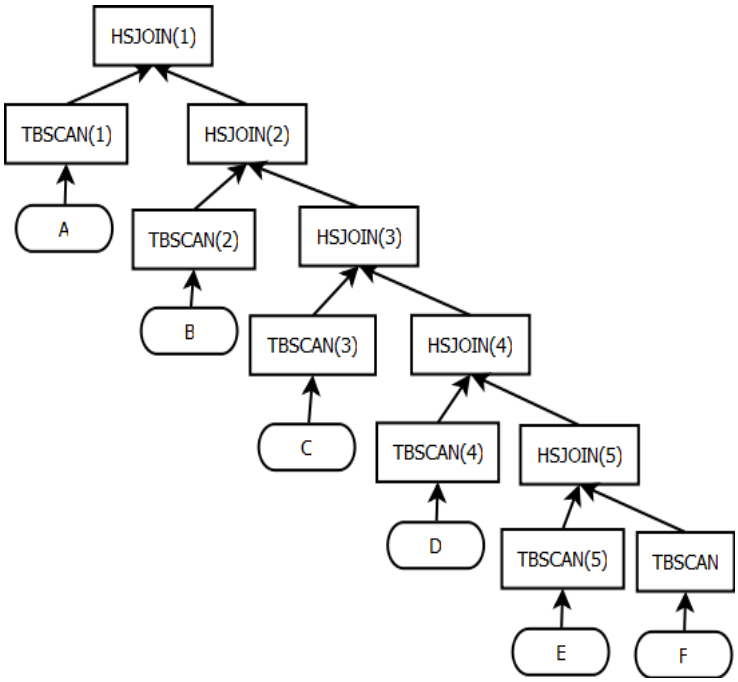


Figure 4.1: Plan Tree Template for Differencing

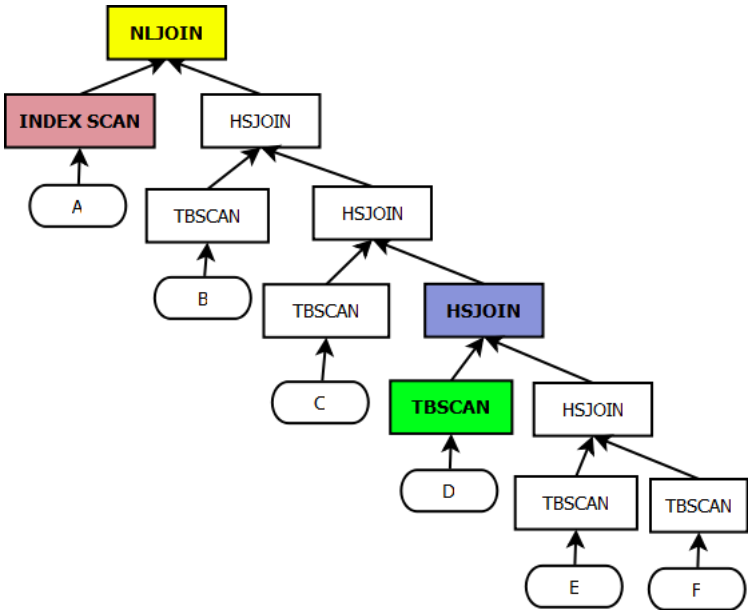
our matching purposes, we do not consider the leaf nodes (relations) in the intersection, since all the plan trees in a given plan diagram have the same set of base relations as their leafs.

Now, ρ is measured as the classical Jaccard Distance [47] between the trees of the two plans, and is computed as

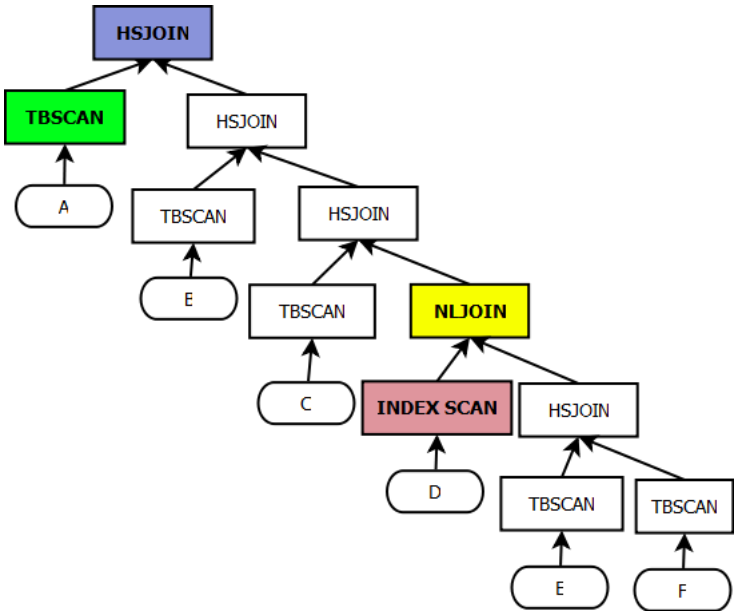
$$\rho(T_i, T_j) = 1 - \frac{|T_i \cap T_j|}{|T_i \cup T_j|} \tag{4.2.1}$$

For example, consider the plan tree template in Figure 4.1. We construct 5 plan trees out of this such that, in the *ith* tree, the nodes labelled *HSJOIN(i)* and *TBSCAN(i)* are re-labelled as *NLJOIN* and *INDEXSCAN*, respectively. Other nodes labelled with *HSJOIN(j)* and *TBSCAN(j)*, where $j \neq i$ are re-labelled to be *HSJOIN* and *TBSCAN*, respectively. With this setting, we can see that between each pair of trees constructed, the value of ρ is $1 - \frac{7}{15} = 0.53$.

Consider the *1st* (Figure 4.2(a)) and *4th* (Figure 4.2(b)) tree instances of the template tree. The colored nodes indicate the differences between the two trees, while white nodes indicate similarity. The leaves of the tree, which are the relations, are not taken into consideration



(a) ***1st*** **tree instance**



(b) ***4th*** **tree instance**

Figure 4.2: Plan tree Differences

while determining differences since they would be present in all plan trees of a plan diagram. So, $|T_i \cap T_j| = 7$ and $|T_i \cup T_j| = 15$. Hence, the value of ρ for these two trees would be $1 - \frac{7}{15} = 0.53$.

Using this method we calculated the plan distances between every two plans, resulting in a matrix of dissimilarities. Note that the ρ values are normalized between 0 and 1, with values close to 0 indicating that all the plans are structurally very similar to each other, and values close to 1 indicating that the plans are extremely dissimilar. Also, the metric is symmetric, with $\rho(T_i, T_j)$ being equal to $\rho(T_j, T_i)$.

We now go on to describe the techniques employed to convert the plan dissimilarities to distances in color space.

4.3 Multidimensional Scaling

As our optimization strategy, we considered methods from the family of multidimensional scaling (MDS) techniques. As described in [7],

MDS: "Multidimensional scaling (MDS) is a method that represents measurements of similarity (or dissimilarity) among pairs of objects as distances between points of a low-dimensional multidimensional space".

The classifications of MDS are provided in [46, 56]. Depending on the level of the input data, different techniques of MDS are recommended in [7]. It is given that if only the rank-order of the dissimilarities (plan tree differences) is considered informative, then *non metric* MDS has to be considered and if the dissimilarities are related to the proximities (color distances) by a specific continuous function, then *metric* MDS techniques need to be employed. Since for our input data (plan tree differences), the dissimilarities are related to the proximities, metric multidimensional scaling is the technique suitable for our purposes.

One of the metric MDS methods discussed in [7] for computing solutions is an iterative algorithm based on *Stress Majorization* called SMACOF (Scaling by majorizing a complicated function). However, due to reasons elaborated later in this chapter, we could not successfully employ this technique even though it has guarantees on the rate of convergence. Instead, we adapt a non-metric MDS technique, which is Kruskal's *Iterative Steepest Descent* [29, 30] (ISD) approach to achieve our goal. The ISD approach is outlined below.

4.3.1 Kruksal's Iterative Steepest Descent

Iterative Steepest Descent: Given a function f for which the minima needs to be determined, we initially assign a random value to kick start the iterative process. In any iteration, we move along the direction of the negative gradient by a chosen step size h. We continue this iterative process till the convergence criteria is met. Usually, the test is to check whether the change in the objective value and/or the value assigned, is more than a small value ϵ.

Choosing the step size (h) carefully is important to monotonically decrease the objective function in every iteration and also for the algorithm to converge quickly. Also, starting at a bad random value might end up in a local minima which is nowhere close to the global minima. To reduce the possibility of this happening, it is recommended in [30] to run the method on a given input with a variety of random positions. We have adopted the recommendation of optimizing with many random inputs and have also carefully chosen h as elaborated in the next section.

4.4 Modelling the Coloring Problem as an MDS Problem

One of the first design considerations before employing MDS would be the number of dimensions in the output space. Since our problem, as defined in Section 4.1 is to convert the dissimilarities into equivalent distances in color, we have the *color* space as our output. This being the case, the **RGB** (Red, Green and Blue), and the **CMYK** (Cyan, Magenta, Yellow, Key - Black) model are among the candidate choices for the output color space. If we choose the CMYK model, when the percentage of the Key (Black) color is 100%, no change in any color by any amount will change the resultant color. This will make a part of the output space redundant and the colors of the vertices placed in that area by the MDS program will not reflect the dissimilarities between them. So, to prevent such situations, we choose the RGB model which makes our output space three dimensional. Thus, we have the mapping

$$g : \mathbb{D} \rightarrow ([0,1]^3)^n$$

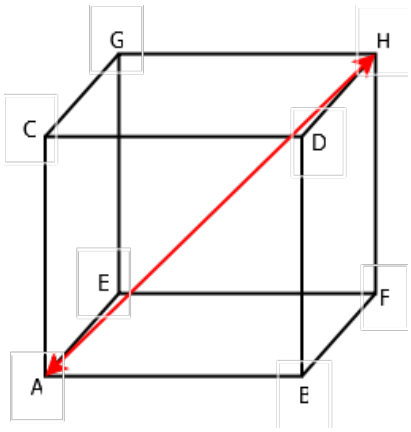


Figure 4.3: Output Color Space

where g represents the scaling function, $\mathbb{D}$ stands for the set of all symmetric matrices of the type $[0-1]^{n*n}$ (i.e., the set of all dissimilarity matrices), one of which is given to us as input, n represents the number of vertices or equivalently, plans in the plan diagram and the three co-ordinates of the $[0,1]^3$ space represent the colors Red, Green and Blue, respectively.

The next design issue is the variant of MDS to use - metric or non-metric, and along with it, the specific technique to be employed. As already mentioned in Section 4.3, we choose a non-metric MDS technique, the ISD approach, as opposed to the popular SMACOF strategy, the reasons for which are elaborated as follows –

We know that all input dissimilarity values are in the range $(0,1]$ and that each of the color values (Red, Green, Blue) are represented in the range $[0,1]$. The latter part necessitates every point in the output of our MDS program to lie in the $[0,1]^3$ range, thus making the output space into an unit cube as shown in the Figure 4.3. But the output of the SMACOF algorithm is in $\mathbb{R}^3$. Even after we translate the output, so that most points are accommodated inside the cube, a large number points still remain outside. Also, if we scale down the values into the cube, it leads to deterioration in the quality of coloring. However, the non-metric ISD (Steepest Descent) algorithm [29, 30], can be easily modified to accommodate the cube constraint. A downside of the adaption is that the convergence rate guarantees provided by classical ISD may no longer hold, but our empirical experience is that even for plan diagrams with hundreds of plans, convergence is achieved within a few minutes.

There is another important feature specific to our coloring problem. As already mentioned, the output space is an unit cube with maximum and minimum values possible being $(0.0, 0.0, 0.0)$ and $(1.0, 1.0, 1.0)$, respectively. The maximum possible distance between any two points in any cube would be along its diagonal, which is shown by a red line joining the vertices A and H in Figure 4.3. Hence in our unit cube, the maximum possible distance is $\sqrt{3}$. But the maximum possible distance in our input space is limited to 1. This would prove to be a handicap since we would confine the output colors to a smaller color space and this may lead to inaccurate representation of distances in the color space. To overcome this drawback, we scaled up all the dissimilarities by a factor of $\sqrt{3}$.

It is important to note that an ideal solution which gives an objective value of zero may not always exist, since scaling down distances to lower dimensions is an inherently lossy procedure. It may not be possible to embed *all* plans in 3D space, such that the distances between them satisfy their respective dissimilarities. As an example, consider the example given in Section 4.2. The dissimilarity matrix corresponding to these 5 plans – P1 to P5 is

$$\begin{pmatrix} 0.0 & 0.53 & 0.53 & 0.53 & 0.53 \\ 0.53 & 0.0 & 0.53 & 0.53 & 0.53 \\ 0.53 & 0.53 & 0.0 & 0.53 & 0.53 \\ 0.53 & 0.53 & 0.53 & 0.0 & 0.53 \\ 0.53 & 0.53 & 0.53 & 0.53 & 0.0 \end{pmatrix}$$

In 3D space, we can have at most 4 points (P1 - P4), to be at equal distances (0.53) from each other as shown in Figure 4.4. There is no means to place the fifth point P5, such that it is at a distance of 0.53 from every point (P1 - P4) so as to obtain ideal coloring. This example clearly shows the infeasibility of obtaining an optimal solution, no matter what the technique.

4.4.1 Iterative Steepest Descent Adapted to the Coloring Problem

The input to the ISD algorithm would be a matrix of dissimilarities, corresponding to every pair of plans. We then compute the following steps iteratively till the change in the value assigned to vertices, is more than a small value ϵ.

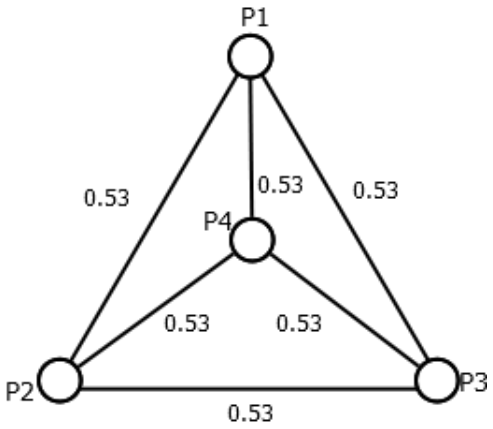


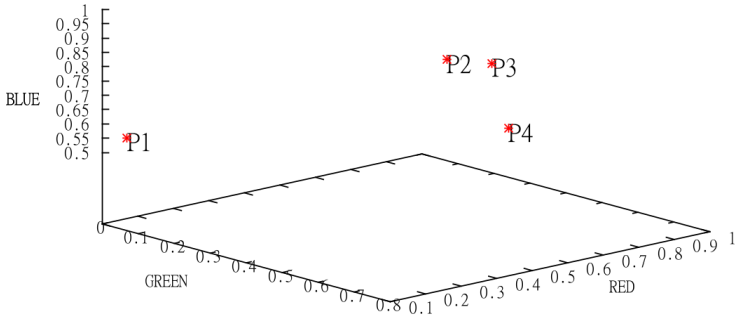
Figure 4.4: Placement of Plans

Computing Slopes. This is done by taking the partial derivative of our objective function at every co-ordinate of every vertex and by adding them up. If any of the co-ordinates of any point is near the border of the unit cube (our output space), then we make its slope to be zero, so that there is no movement along that co-ordinate.

Step Size. After the gradient is computed, we need to calculate the step size with which the co-ordinates of the vertices should be moved. We choose the step size as the minimum of the maximum distance any co-ordinate of any vertex can move inside the cube, along its negative gradient.

Move the vertices. We move the vertices along their negative gradient (in all three dimensions) with the step size decided. If the objective value increases, then we halve the step size and again move the vertices. We continue this procedure till a step size that decreases the objective value is found or no move can be made in this iteration.

As an example consider the following dissimilarity matrix for a set of 4 plans P_1, P_2, P_3 and P_4, with $\epsilon = 0.01$.



(a) Initial Random Configuration

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 0.0 & 0.825 & 0.913 & 0.820 \\ 0.825 & 0.0 & 0.944 & 0.807 \\ 0.913 & 0.944 & 0.0 & 0.270 \\ 0.820 & 0.807 & 0.270 & 0.0 \end{pmatrix} \end{array}$$

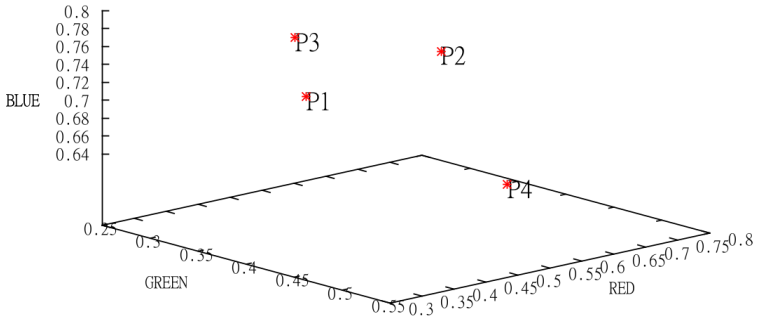
(b) Initial Dissimilarities: Objective =1.335

Figure 4.5: Initial Configuration and Dissimilarity Matrix

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 0.0 & 0.730 & 0.240 & 0.550 \\ 0.730 & 0.0 & 0.637 & 0.597 \\ 0.240 & 0.637 & 0.0 & 0.333 \\ 0.550 & 0.597 & 0.333 & 0.0 \end{pmatrix} \end{array}$$

Figure 4.5(a) graphically shows the initial random locations that are assigned to the 4 plans. The dissimilarity values, and the corresponding objective value obtained from this assignment is given in Figure 4.5(b).

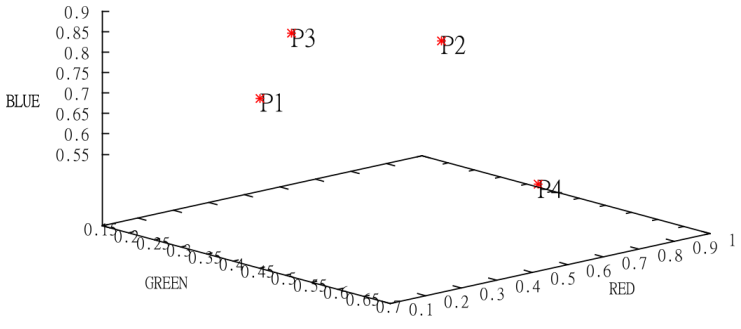
The working of Kruskal's Iterative Steepest Descent method on these initial set of values is shown graphically in Figure 4.6. Figures 4.6(a) and 4.6(b) correspond to the configuration obtained after one iteration of the algorithm. We can see that the objective value has come down to 0.714. After iteration 2, it further comes down to 0.139. After 10 iterations, which corresponds to Figures 4.6(g) and 4.6(h), the algorithm converges with a low objective value of 3.4E-4, since no vertex (plan) moved by more than 0.01 units from its previous position. In any given iteration, it can be observed that each vertex is moved by a value h (step-size), in one, two or all three dimensions.



(a) After 1 Iteration

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 0.0 & 0.438 & 0.077 & 0.207 \\ 0.438 & 0.0 & 0.393 & 0.358 \\ 0.077 & 0.393 & 0.0 & 0.230 \\ 0.207 & 0.358 & 0.230 & 0.0 \end{pmatrix} \end{array}$$

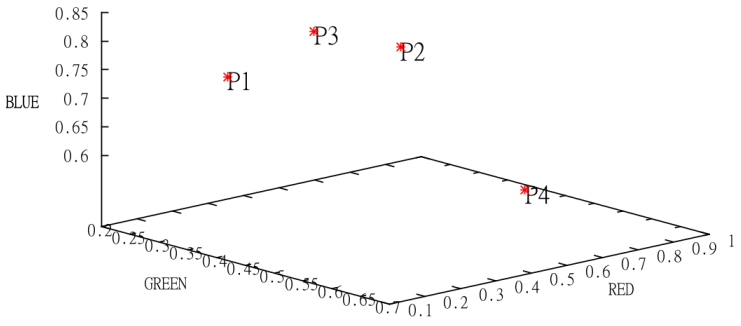
(b) Dissimilarity Matrix: Objective = 0.714



(c) After 2 Iterations

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 0.0 & 0.907 & 0.232 & 0.524 \\ 0.907 & 0.0 & 0.719 & 0.676 \\ 0.232 & 0.719 & 0.0 & 0.491 \\ 0.524 & 0.676 & 0.491 & 0.0 \end{pmatrix} \end{array}$$

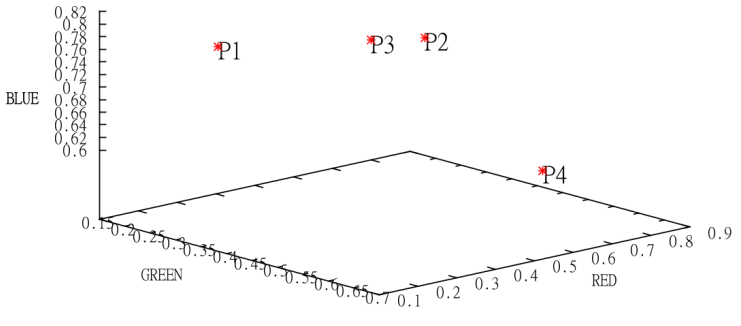
(d) Dissimilarity Matrix: Objective = 0.139



(e) After 3 Iterations

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 0.0 & 0.820 & 0.231 & 0.519 \\ 0.820 & 0.0 & 0.621 & 0.597 \\ 0.231 & 0.621 & 0.0 & 0.390 \\ 0.519 & 0.597 & 0.390 & 0.0 \end{pmatrix} \end{array}$$

(f) Dissimilarity Matrix: Objective = 0.02



(g) After 10 Iterations

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} P_1 & P_2 & P_3 & P_4 \end{array} \\ \begin{pmatrix} 0.0 & 0.727 & 0.241 & 0.541 \\ 0.727 & 0.0 & 0.630 & 0.593 \\ 0.241 & 0.630 & 0.0 & 0.335 \\ 0.541 & 0.593 & 0.335 & 0.0 \end{pmatrix} \end{array}$$

(h) Dissimilarity Matrix: Objective = 3.4E-4

Figure 4.6: Working of Kruskal's ISD

4.5 Experimental Results and Analysis

In this section we color the plan diagrams by employing the technique developed in the previous sections. Our test-bed is the same as that described in the previous chapter (Chapter 3), with QT2, QT8 and QT9.

While experiments were conducted with a variety of weight assignments as discussed in Section 4.1, in this section, we elaborate on the first case where $w_{ij} = w_{v_i} + w_{v_j}$, giving preference to plan pairs with larger total area,

In the following discussion, we refer to 'pairs of plans being treated well', and by that we mean that the ratio of the color distance between any two plans to the Jaccard dissimilarity between them is between 0.7 and 1.3. (error tolerance level is 30%). We choose 30% as our error threshold in order to utilize the coarseness of color perception in the human eye, which cannot distinguish between two closely placed colors in the color spectrum. In the case of minimizing our objective function with weights being $w_{ij} = w_{v_i} + w_{v_j}$, each of the plans is associated with the area covered by the plan in the plan diagram. Hence, it is perhaps reasonable to look at the weighted set of plan pairs treated well. The quantity '% of points pairs treated well' gives the sum of weights of the set of plan pairs treated well divided by the total sum of weights of the plan pairs. Naturally, like the weights assigned in the objective function, the weight given to a plan pair is the sum of the areas of the two plans.

4.5.1 Results

In the process of analyzing the semantically colored plan diagrams, let us first look at the distribution of Jaccard dissimilarities between plans in the plan diagrams of all three templates for both engines. These graphs are shown in Figures 4.7 and 4.8 for Opt A and Opt B, respectively. It is very evident that for Opt A, in all three templates, we have a large number of plans (80, 128, 114), along with conspicuous peaks at higher Jaccard dissimilarities (> 1), indicating that, with our adopted metric, most plans are highly dissimilar from each other. This is in sharp contrast to Opt B for the cases of QT2 and QT8, where we not only have few plans in comparison (15, 40), but also have peaks in the lower dissimilarity range, around 0.5 for QT2 and 0.75 for QT8.

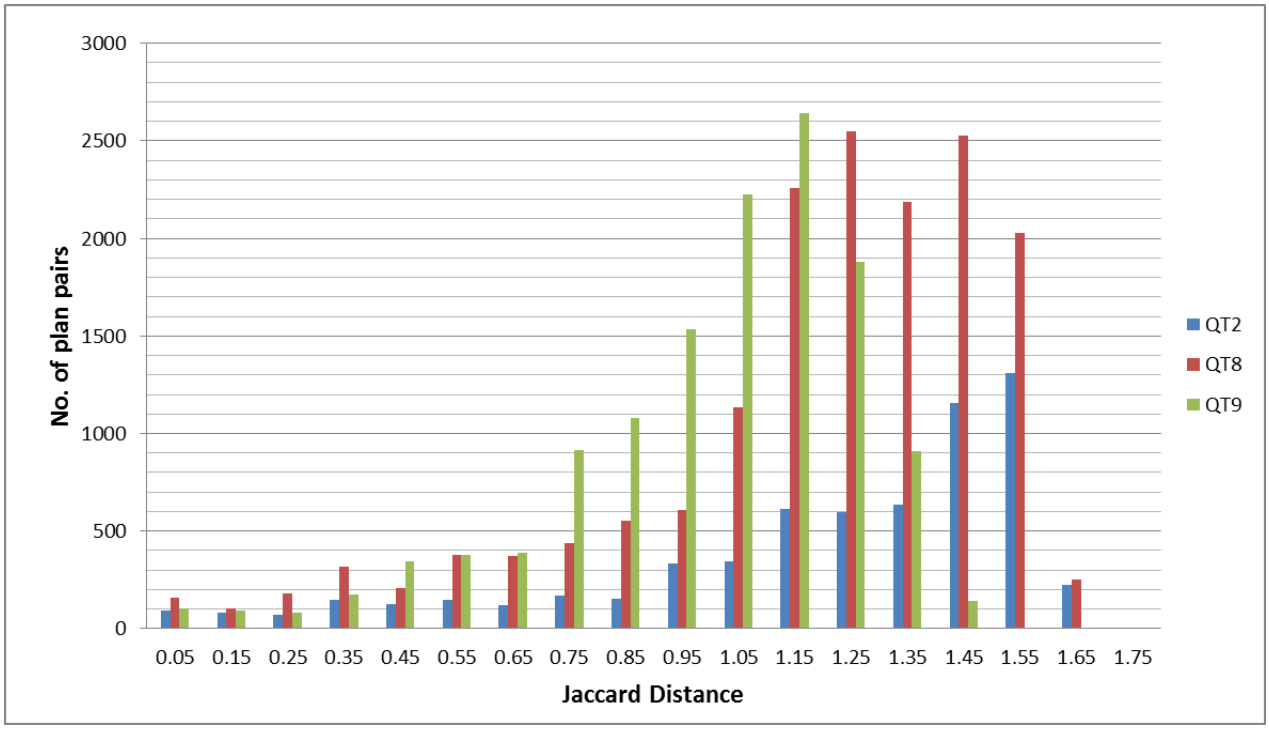


Figure 4.7: Distribution of Jaccard distances for Opt A (scaled to $\sqrt{3}$)

However, QT9 - Opt B does have high plan cardinality and a large dissimilarity range, along with a peak at the Jaccard distance of 1.05.

Now with these numbers in mind, it would not be wrong to expect plan diagrams that are largely differently colored for all templates of Opt A and for QT9 - Opt B, and with fewer colors for QT2 and QT8 of Opt B. The semantically colored plan diagrams of Opt A and Opt B are shown in Figures 4.9 and 4.10, respectively.

For the semantically colored plan diagram of QT2 - Opt A, shown in Figure 4.9(a), we expected a plan diagram with large color variations, and hence the coloring obtained is surprising. A large part of the plan diagram has shades of red along with maroon and orange. The explanation for this is that a large area of the diagram is occupied by very few plans, this being evident by the high Gini co-efficient of 0.79. Also, large number of plans are found concentrated near the axes, and these plans do have colors which are far apart in color space. We have plans P23 and P67, colored bright yellow and blue (color distance of 1.73), respectively, both being thin strips near the axes. This would be one of the diagonals in our cube. Also, there is the (P21,

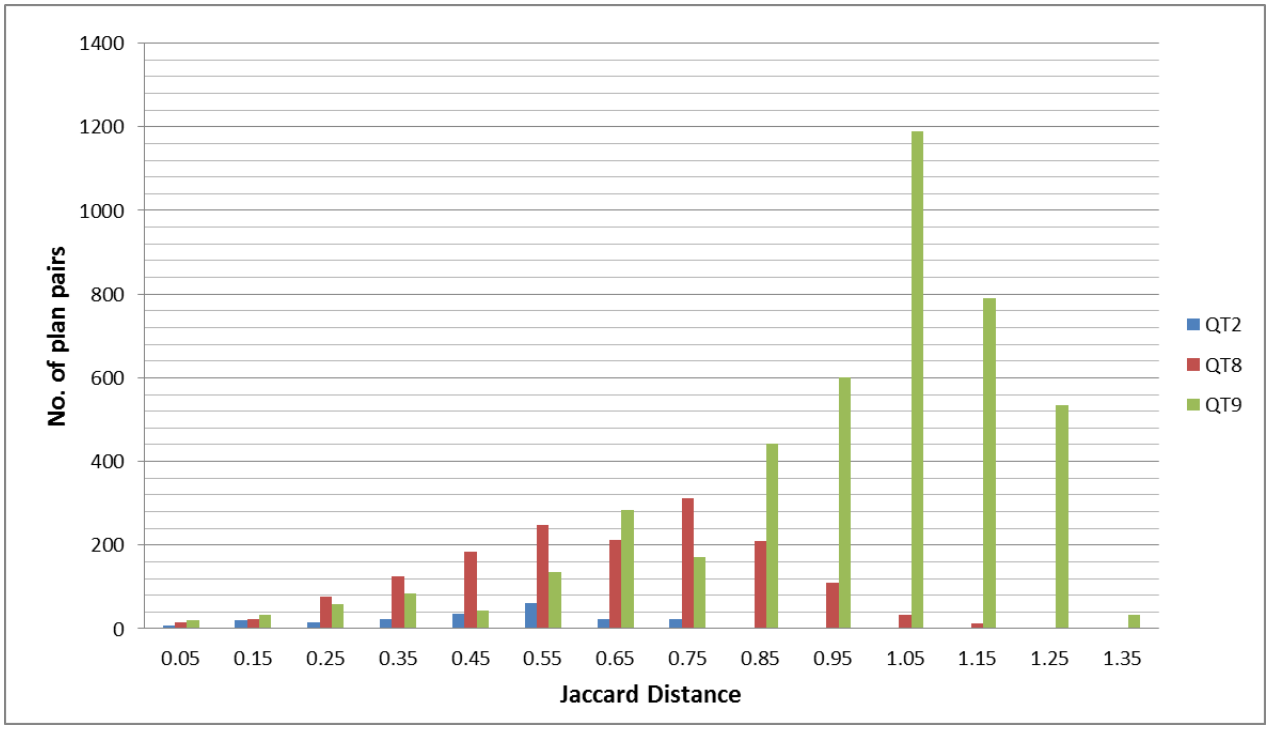
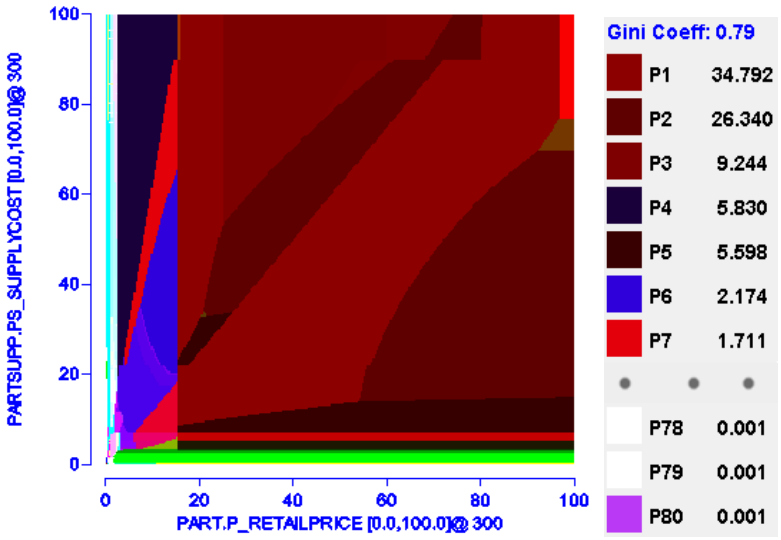


Figure 4.8: Distribution of Jaccard distances for Opt B (scaled to $\sqrt{3}$)

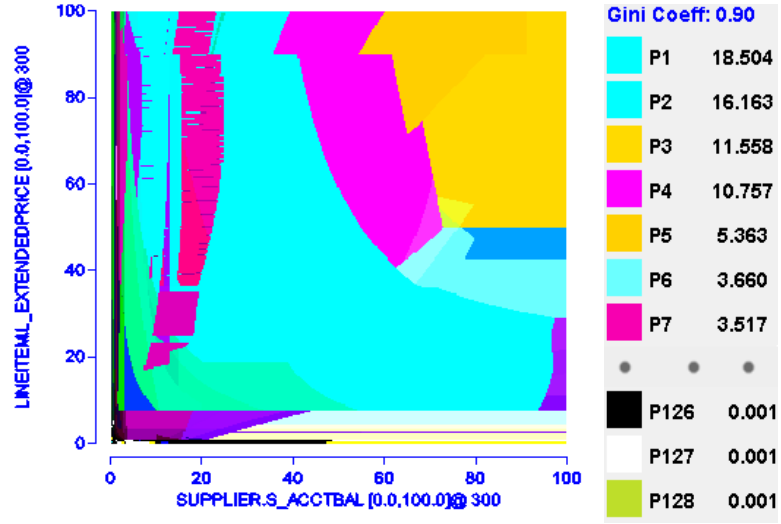
P48) pair colored (bright green, pink) that are situated in the ends of another diagonal in the cube. However, the maximum Jaccard distance found in this plan diagram corresponds to the plan pair (P24, P44) with 1.66.

With respect to the semantically colored plan diagram of QT2 - Opt B, shown in Figure 4.10(a), the coloring approximately matches our prediction, with bluish-green shades in the upper half of the diagram. The maximum difference in color in this plan diagram is seen between P12 and P15 with 0.981 (actual dissimilarity is 0.356). These plans correspond to thin vertical bars very close to Y-axis. The maximum Jaccard dissimilarity of 0.794 is found for the pair (P3, P10). They correspond to the light orange squarish patch located in the centre of the bottom part, and the dark green vertical strip, situated leftmost in the upper half of the plan diagram. The color dissimilarity corresponding to this pair is 0.781.

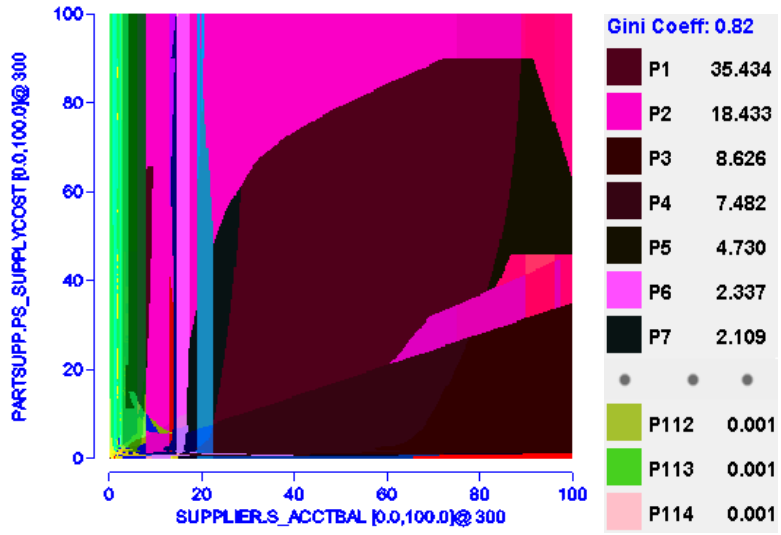
Moving to QT8 - Opt A, we see behavior similar to what we saw in QT2 - Opt A. We have a relatively large number of plans (128), and a rather flat peak in the dissimilarity range $[1, 1.5]$ indicating that there a large number of plans which are highly dissimilar from each other, but a



(a) **QT2 - Opt A**

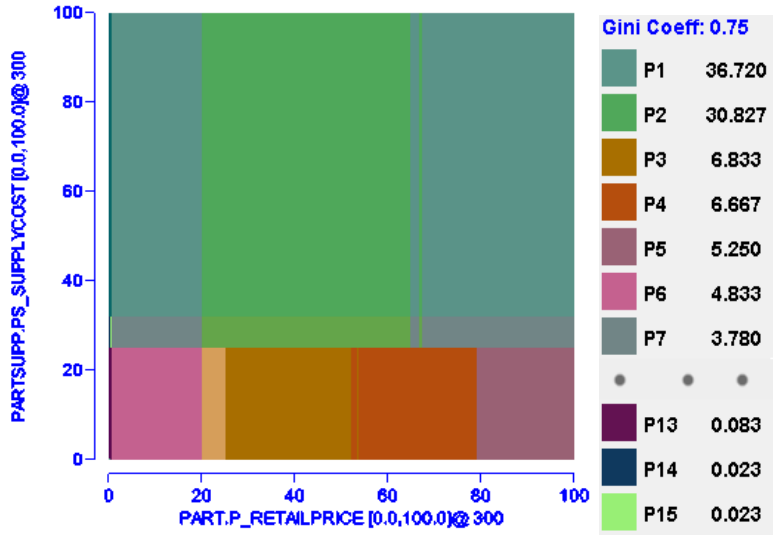


(b) **QT8 - Opt A**

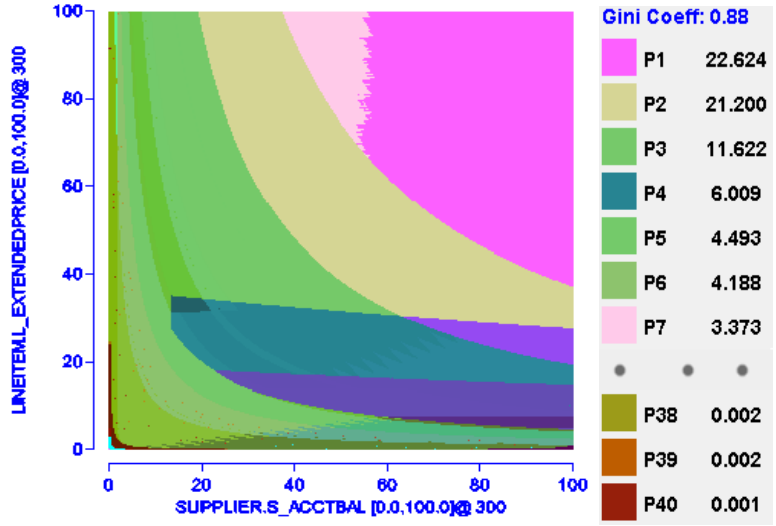


(c) **QT9 - Opt A**

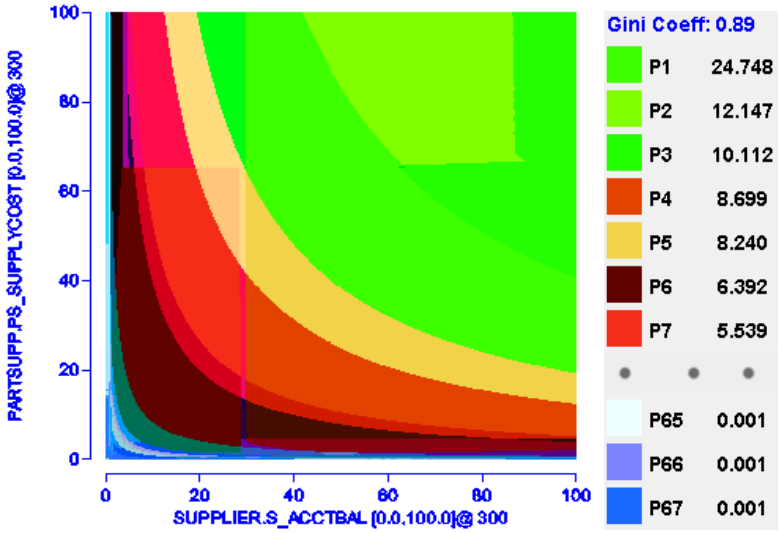
Figure 4.9: MDS Colored Plan Diagrams for Opt A



(a) **QT2 - Opt B**



(b) **QT8 - Opt B**



(c) **QT9 - Opt B**

Figure 4.10: MDS Colored Plan Diagrams for Opt B

QT	No of plans	% of point pairs treated well	% of plan pairs treated well	Lowest ratio	Highest ratio	Objective with weights	Objective w/o weights
2	80	89%	66.8%	0.0	776.5	0.03	0.22
8	128	87.9%	62%	0.02	381	0.06	0.27
9	114	93.1%	62.5%	0.0	722	0.01	0.15

Table 4.1: Statistics for ISD With Weights for Opt A

large part of the plan diagram is colored with cyan and light green – colors which are close to each other. The maximum color and Jaccard distances are 1.73 and 1.70 corresponding to plan pairs (P126, P127) and (P22, P117), respectively. Plans P126 and P127 are colored black and white, respectively and occupy too small a area to be noticed by the eye in the plan diagram.

QT8 - Opt B, has a smaller dissimilarity range (0 - 1.1) as compared to QT8 - Opt A. We can observe from Figure 4.10(b), the semantic colored plan diagram of QT8 - Opt B, that starting from the origin, till about the middle of the plan diagram, plans are colored with similar shades of green and blue. This suggests that the plans in the plan diagram are largely similar till the selectivity crosses 25% along both X and Y axes. The maximum Jaccard distance is 1.15 (color distance 1.18) between P1 and P23, P1 being colored pink at the top right and P23 being the dark brown colored plan near the Y-axis. The maximum color distance is 1.47 between plans P28 and P29, both of which are found very close to the axes, with Jaccard distance between them being 0.001. They occupy less than 0.025% of the plan diagram space, which explains why their color distance is very far apart from their actual dissimilarity value.

Inferences similar to the previous two cases can be drawn regarding the coloring of plan diagram for both QT9 - Opt A and QT9 - Opt B. One observation regarding all plan diagrams is that the maximum color distance achieved in each of them is slightly higher than the corresponding maximum Jaccard distance found.

The accuracy of the interpretations that can be drawn from the MDS colored plan diagrams do not seem far from reality when we look at the statistics of the generated coloring in Tables 4.1 and 4.2.

The objective values achieved in all the cases are sufficiently low to indicate that the plan diagrams are reasonably well colored. Further, to justify this claim we can see that the ‘% of

QT	No of plans	% of point pairs treated well	% of plan pairs treated well	Lowest ratio	Highest ratio	Objective with weights	Objective w/o weights
2	15	98.6%	84%	0.45	7.77	0.002	0.025
8	40	96.0%	76.0%	0.04	1362.0	0.01	0.044
9	67	92.5%	57.1%	0.01	173	0.02	0.124

Table 4.2: Statistics for ISD With Weights for Opt B

point pairs treated well', shown in column 3, is more than 85% for all three templates. '% of plan pairs treated well', given in column 4, is considerably less in many cases (around 60%). This low percentage can be attributed to the fact that we have tried to optimize the objective function with weights being the areas of the plans. Because of this, plans with larger areas tend to be colored well, and since we have a high skew (Gini Co-efficient of 0.79), a large part of the plan diagram will be occupied by very few plans, and those few plans tend to be colored well at the cost of many other plans (which occupy very less area) not being colored within our error tolerance range.

Also, one can notice the increase in the percentage of '% of point pairs treated well' as the value of the objective function decreases. Even though our objective function directly does not optimize the '% of point pairs treated well', it has the necessary effect in our situation.

The objective function value calculated without weights is more than the value obtained when it is calculated with weights, as we can see in columns 7 and 8. This is expected since the optimization was carried out with weights. The columns highest ratio and lowest ratio provide information about the worst case performance of the technique employed. We can see that the ratios can get as bad as possible, like in the case of QT9 - Opt A, where the lowest ratio is as low as 0.

4.5.2 Time Overheads of ISD

In Table 4.3, we have tabulated the time taken and the number of iterations required to run ISD for numerous query templates. All values have been computed with a convergence criteria of $\epsilon <= 1.0E-5$.

We observe here that, in general the average time taken increases with the increase in the

QT	No. of plans	Avg. No. of Iterations	Max. Iterations	Min. Iterations	Avg. Time Taken (in seconds)
QT2-OptB	15	386	814	229	<1
QT8-OptB	40	4465	16807	2315	5.6
QT9-OptB	67	2669	6851	616	8
QT2-OptA	80	1927	3712	20	7.7
QT8-OptA	128	1435	3374	675	13
QT9-OptA	114	6426	16593	2668	59.4

Table 4.3: Time Overheads of ISD

cardinality of the plan diagram. For QT9-OptA, however, which has lower plan cardinality than QT8-OptA, the average time taken is much higher than QT8-OptA. This is because QT9-OptA requires larger number of iterations as compared to QT8-OptA in order to meet the convergence criteria. These large number of iterations are because there are a set of plans that are equally dissimilar from each other, and in the process of trying to maintain their dissimilarities in the color space, the algorithm keeps moving the plans back and forth in successive iterations, with very small changes in the objective value.

4.6 Representational Quality of Iterative Steepest Descent

From the discussion on experimental results in the previous section, a natural question that would surface is whether we could do better in terms of the representational quality by the use of a different technique. Representational quality includes '% of point pairs treated well', '% of plan pairs treated well' and the worst case ratios obtained. In this section, we compare the results obtained through our adapted ISD with those obtained from SMACOF, and show empirically that the performance is largely comparable.

The statistics obtained from coloring plan diagrams of the same three templates – QT2, QT8 and QT9, with SMACOF, under conditions similar to that of ISD are shown in Tables 4.4 and 4.5 for Opt A and Opt B, respectively.

We can see that the values in columns 3 and 4, corresponding to '% of point pairs treated well' and '% of plan pairs treated well' are only slightly better than those obtained from ISD.

QT	No of plans	% of point pairs treated well	% of plan pairs treated well	Lowest ratio	Highest ratio	No. of points outside cube
2	80	96.3%	72.9%	0.05	321.5	45
After Scaling		23.12%	25.72%	0.0	294	0
8	128	92.8%	63.7%	0.03	634	81
After Scaling		51.3%	50.86%	0.0	634	0
9	114	94.71%	64.3%	0.007	906	66
After Scaling		59.77%	55.27%	0.003	588.9	0

Table 4.4: Statistics for SMACOF With Weights for Opt A

QT	No of plans	% of point pairs treated well	% of plan pairs treated well	Lowest ratio	Highest ratio	No. of points outside cube
2	15	98.7%	86%	0.43	1.92	0
After Scaling		98.7%	86%	0.43	1.92	0
8	40	96.76%	77.4%	0.07	19.18	1
After Scaling		96.76%	77.4%	0.07	19.18	0
9	67	93.9%	62.2%	0.04	157	20
After Scaling		82.1%	55.22%	0.04	157	0

Table 4.5: Statistics for SMACOF With Weights for Opt B

However, the numbers corresponding to the lowest and highest ratios are better in SMACOF. But, these results from SMACOF are currently not bounded to a cube, and the number of points outside the cube, as given in the last column of the tables, is high in many cases. For the case of Opt A, more than half the plans are outside the cube in the case of all three templates. The statistics obtained after scaling down the values are given just below the ones obtained before scaling. We can clearly see that both "% of plan pairs treated well" and "% of point pairs treated well" deteriorate, sometimes by an order of magnitude. Armed with the above facts, we conclude that the ISD technique colors plan diagrams satisfactorily.

Chapter 5

XML Plan Diagrams

In recent years, the hierarchically-structured XML document format has found its way into mainstream database technology, and several popular database systems have been extended and developed to support XML storage and XML query processing along with relational data. To this effect, there has been considerable interest in the past decade for optimizing XML queries and in particular XQuery, which has been the W3C recommended standard for querying XML [65].

A lot of effort has been invested in the recent past for analyzing the behavior of SQL optimizers. As discussed in Chapter 1, the notion of *plan diagrams* has become widely popular as a tool for such analysis. In the previous two chapters we explored two new semantic features of plan diagrams which help analyze the behavior of SQL optimizers better. In this chapter we focus on analysing the behavior of one such XQuery optimizer, referred to as DB_XML in the way it handles XQuery, through the production of XML plan diagrams.

Generation of plan and other related diagrams for DB_XML comprised of the following main tasks:

XML Statistics Collection: Mechanisms for generation and storage of statistics on XML data in DB_XML had to be first identified. Then, methods had to be devised to access this information.

XQuery Templates: The format of an XQuery template had to be defined and constraints for a valid XQuery template had to be identified.

XML Selectivity Computations: Mechanisms for identifying, interpreting and computing selectivities from the statistical summaries and the optimizer's plan output had to be designed.

Execution Plan Parsing: All new operators introduced in DB_XML to handle XQuery had to be taken into account while identifying trees in the process of production of plan diagrams.

In this chapter, we first describe in detail each of the above steps. Then we provide extensive experimental analysis on DB_XML's behavior using a set of three benchmarks. Further, we discuss the results obtained by applying the structure-based reduction technique from Chapter 3. Finally, we present and analyze XML plan diagrams that are semantically colored using techniques developed in Chapter 4

5.1 XML Statistics Collection

In the relational world, statistics collection is at the granularity of individual attribute columns, with all relevant information stored in internal system tables, which can be directly accessed through SQL queries. For XML, however, statistical information is organized with respect to *element paths* (e.g. `/Customers/Customer/AccountBalance`), and the storage and access of the information is considerably more involved, as explained next.

We will hereafter use the term XML_R to refer to a database relation that has one or more XML columns (columns that have XML data). Given an XML_R, executing the runstats utility (an utility provided by the DB_XML engine) on this relation provides the LOW2KEY and HIGH2KEY values computed over all paths in the XML documents referenced by this relation, as well as the total number of such instances. Further, to obtain distributional information, the runstats utility has to be invoked using the *"with distribution"* option – a special requirement in DB_XML is that this option is effective only after an *index* has been created on the associated path.

The statistics generated for the XML_Rs corresponding to the element paths that define the dimensions of the plan diagram, are dumped into a text file using another utility provided

by the DB_XML engine. Due to the lack of utilities for processing this file, we wrote our own Perl parser to analyze the contents. The extracted information is stored in two tables, XML_SUMMARY and XML_HIST, whose schemas are shown in Figure 5.1.

XML_SUMMARY (colid bigint, tbname varchar(50), elementpath long varchar, card bigint, high2key varchar(100), low2key varchar(100))

XML_HIST (indexid bigint, indexpath long varchar, colid bigint, tbname varchar(50), dtype varchar(32), value varchar(100), valcount bigint, distinctcount bigint)

Figure 5.1: Tables for storing statistics information

The XML_SUMMARY table contains the low keys, high keys and cardinalities of the various element paths, whereas the XML_HIST table contains the histogram values and data types for the terminating elements of the indexed paths. The information is separated over two tables because the utility dumps statistics for *all paths* of XML documents stored in a XML column, some of which may have an index, while others may not, in which case they would lack distributional information. Further, even in the presence of indexes, the information present in XML_SUMMARY would be repeated as many times as the number of buckets in the histogram summary, violating normalization imperatives.

To make the above notions concrete, consider an example bank customer database, where the XML nodes have the structure shown in Figure 5.2(a), consisting of the customer's account number, name and balance. Sample summary statistics and a snippet of the (equi-depth) histogram on the acct element path `(/root()/customer/acct/text())`, as produced by DB_XML, are shown in Figures 5.2(b) and 5.2(c), respectively.

5.2 XQuery Templates

The relational columns that form the dimensions of SQL-based optimizer diagrams are called *Variable Selectivity Predicates (VSP)*. These VSPs are easily specified using the :varies syntax embedded in SQL, as previously shown in Figure 1.3. With XQuery, however, the specification is more complicated since the dimensions of the optimizer diagrams are now *element paths* in

<customer>
< acct> 1881 < /acct>
< name> Pablo Picasso < /name>
< bal> 1000 < /bal>
< /customer>

(a) XML node

PathID = /root()/customer/acct/text()
2nd Highest Key = 5:6059
2nd Lowest Key = 3:01
Sum Node Cnt = 3721

(b) Summary Stats

Value	Count	Distinct Count
0	1	1
12	16	13
19	30	20
27	46	28
34	60	35
42	76	43
49	90	50
57	106	58
68	120	69
82	134	83
...	...	...

(c) Frequency Histogram

Figure 5.2: Example XML node and Statistics

the XML hierarchy, and it is difficult to directly infer these paths from the :varies predicate in the query. In particular, the complete specification of an optimizer diagram dimension requires information on the path structure, namespaces, table names, aliases and column names. We will heareafter use the term **VSX** (Variable Selectivity XML element path)[1] to denote these dimensions and distinguish them from the relational VSP notation.

All information about a particular VSX is provided as a *prologue* to the query template, as shown in Figure 5.3, where there are two VSXs corresponding to customer accounts and orders. This XQuery template retrieves those accounts that have a working balance within a parametrized value and, for each of these accounts, extracts the associated orders involving cash transactions within a parametrized value.

Here, the namespaces corresponding to the XQuery template are enumerated in the begin-

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

```
#NAMESPACE declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4";
#NAMESPACE declare namespace c="http://tpox-benchmark.com/custacc";

#VSX /c:Customer/c:Accounts/c:Account/c:Balance/c:WorkingBalance
CUSTACC CUSTACC CADOC
#VSX /FIXML/Order/OrdQty/@Cash ORDER ORDER ODOC
# – #

XQUERY
declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
for $acct in db2-fn:xmlcolumn("CUSTACC.CADOC")
   /c:Customer/c:Accounts/c:Account[c:Balance/c:WorkingBalance :varies]
   for $ord in db2-fn:xmlcolumn("ORDER.ODOC")
      /FIXML/Order[@Acct=$acct/@id/fn:string(.)]
      where $ord/OrdQty/@Cash :varies
      return
         <result>
           {$acct}
           {$ord}
         </result>
```

Figure 5.3: Example XQuery Template

ning of the prologue followed by VSXs. Each VSX in turn consists of its *element path*, *table name* (i.e. its XML_R), *alias* and *column name* (i.e. the XML column), listed in that order. The beginning of each namespace and each VSX is indicated through the use of #NAMESPACE and #VSX, respectively, and the termination of the prologue is indicated through #–#. Note that the :varies keywords are still embedded inside the Xquery template so as to unambiguously determine where values would be substituted for varying the VSX. As a final point, the order in which the VSX locations are listed in the prologue should match the lexical order of the corresponding :varies predicates within the XQuery template.

Template Constraints. In order to meaningfully cover the full range of selectivities shown in the various optimizer diagrams, an XQuery template should satisfy a variety of conditions, which are enumerated below (for reference, the equivalent constraint in the SQL framework is

also listed):

1. Each XML column of a relation can participate in at most one VSX. (In SQL templates, each *relation* can participate in at most one VSP.)

2. 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 in the query template. To make this concrete, consider the VSX `/c:Customer/c:Accounts/c:Account/c:Balance/c:WorkingBalance` in the XQuery template of Figure 5.3. Here L is the segment (`/c:Customer/c:Accounts/c:Account`), while P is the downstream segment `/c:Account/c:Balance/c:WorkingBalance` – the parametrized variation across Customer Accounts is achieved through the varying of WorkingBalance values. Similarly, for the second VSX, the logical segment is `/FIXML/Order` and the physical segment is `/Order/OrdQty/@Cash`. 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.

   For a VSX predicate with an explicit P segment, it should be ensured that *many-to-one* relationships do not occur in this segment. That is, in the graphical representation of the XML schema [43], there should be no '*' node in the path corresponding to P. (In SQL templates, there is no equivalent to these notions of logical and physical segments since hierarchies are inherently not present.)

3. If along with the VSX predicate, other predicates are also specified, then it should be ensured that the application of these additional predicates is deferred in the plan tree. Specifically, these non-VSX predicates should not be applied in conjunction with the VSX predicate, at the leaf level, in the plan tree. (In SQL templates, VSP relations can feature in join predicates but not in any other equality or range predicates.)

4. The permissible data types for VSX element paths are *integer*, *float*, *string* and *date*. (The same set of data types are supported in SQL templates.)

5. The VSX element paths must have pre-generated statistical summaries. (The same constraint holds for VSPs in SQL templates.)

6. The VSXs should be on dense-domain paths in high cardinality XML columns. (The same constraint holds for VSPs in SQL templates.)

To illustrate the constraints on XQuery templates, consider an XML database with CUSTOMER and ORDER documents adhering to the schema graphs shown in Figures 5.4(a) and 5.4(b), respectively. On this database, consider the XQuery template shown in Figure 5.4(c), which returns the names and phone numbers of customers located within a parametrized address value and whose orders feature item quantities within a parametrized value – this template is compatible with respect to all of the above conditions.

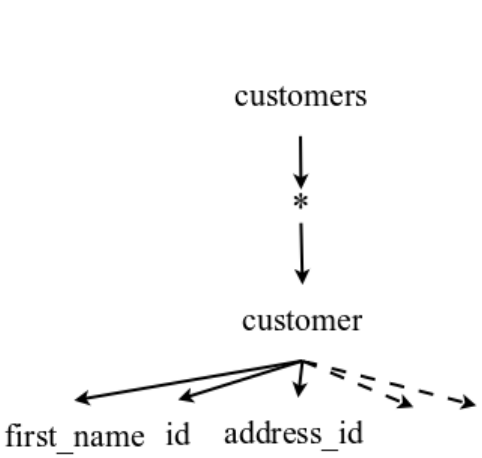
However, if the query template were to be slightly altered as shown in Figure 5.4(d), it would become invalid as Condition 2 of the above requirements would be violated – specifically, there is now a '*' node in the physical segment (`order_lines/('*')order_line/quantity_of_item`).

5.3 XML Selectivity Computation

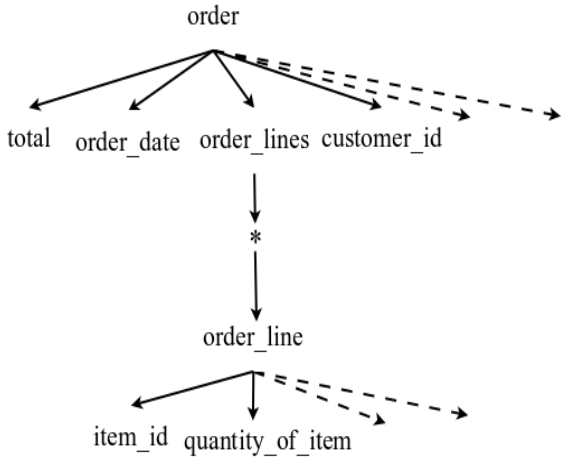
After the statistics are collected and the information regarding VSXs are retrieved from the XQuery template, the next step in obtaining plan diagrams from DB_XML is to determine the constants that would result in the desired VSX selectivity values. In the following discussion, we first explain the selectivity computation process in the relational environment, and then describe its extension to the XML world.

5.3.1 Relational Selectivities

As mentioned earlier, in the process of producing the plan diagram, we estimate the constants that would result in the desired selectivities through an "inverse-transform" of the VSP statistical summaries. To verify that the estimates provided by us and the optimizer are mutually consistent, three types of selectivities are computed: *Expected Selectivity*, *Predicate Selectivity*



(a) CUSTOMER Schema Graph



(b) ORDER Schema Graph

```
#PSX /customers/customer/address_id CUSTOMER CUSTOMER CUSTOMER
#PSX /order/order_lines/order_line/quantity_of_item ORDER ORDER ORDER
#-#
XQUERY
for $cust in db2-fn:xmlcolumn("CUSTOMER.CUSTOMER")
 /customers/customer[address_id :varies]
 for $order in db2-fn:xmlcolumn("ORDER.ORDER")
  /order[customer_id=$cust/@id]/order_lines/order_line[quantity_of_item :varies]
   return <Output>
     {$order}
     {$cust/first_name}
     {$cust/last_name}
     {$cust/phone_number}
   </Output>
```

(c) Valid XQuery Template

```
#PSX /customers/customer/address_id CUSTOMER CUSTOMER CUSTOMER
#PSX /order/order_lines/order_line/quantity_of_item ORDER ORDER ORDER
#-#
XQUERY
for $cust in db2-fn:xmlcolumn("CUSTOMER.CUSTOMER")
 /customers/customer[address_id :varies]
 for $order in db2-fn:xmlcolumn("ORDER.ORDER")/order[customer_id=$cust/@id
  and order_lines/order_line/quantity_of_item :varies]
   return <Output>
     {$order}
     {$cust/first_name}
     {$cust/last_name}
     {$cust/phone_number}
   </Output>
```

(d) Invalid XQuery Template

Figure 5.4: XQuery Template Constraints

and *Plan Selectivity*. These selectivities, whose values are expected to be close to each other in a valid execution, are defined as follows:

Expected Selectivity: This is the selectivity of the VSP as determined by us through the statistical summaries of the database engine.

Predicate Selectivity: This is the optimizer's estimated selectivity for the VSP when the *unidimensional* query

*select * from Table T where VSP(T)*

is optimized.

Plan Selectivity: This is the selectivity associated with the node containing the VSP in the optimizer's plan choice for the user's query.

5.3.2 XML Selectivities

For the above computations in the relational world, selectivity is simply and consistently interpreted as the fractional number of rows in a table that are relevant to processing the user's query. However, with XML data, the definition is not so straight-forward since information is organized in the form of *nodes* and *documents* containing these nodes. Therefore, selectivities can be computed at the granularity of nodes or 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.

From a query processing perspective, *node selectivities* constitute a more meaningful metric than document selectivities since they essentially correspond to rows in tables. Unfortunately, the current implementation of indexes in DB_XML is such that although information to compute node selectivities is available, only *document* selectivities are finally returned as outlined in [4]. As a further complication, the document selectivities computed at the leaves of the plan trees later *morph* into node selectivities at the higher internal levels of the plan tree.

5.3.3 Computation Methods

Modulo the above conceptual issues, the Predicate and Plan selectivities of the DB_XML optimizer are computed in the following manner:

Predicate Selectivity: The representative XQuery structure shown in Figure 5.5(a) is optimized, with its place holders – `NAMESPACES`, `TABLE_NAME`, `COLUMN_NAME`, `VSX` and `constant-value` – filled up using the data from the XQuery template under consideration. The Predicate Selectivity is taken to be the estimated number of XML results in the final output, i.e., the number present as output in the RETURN operator of the plan tree returned by the optimization process.

```
XQUERY
NAMESPACES
for $x in
  db2-fn:xmlcolumn("TABLE_NAME.COLUMN_NAME")VSX
  where $x <= constant-value
  return $x
```

(a) XQuery Structure

```
XQUERY
declare default element namespace
  "http://www.fixprotocol.org/FIXML-4-4";
declare namespace c="http://tpox-benchmark.com/custacc";
for $x in
  db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order/OrderQty/@Cash
  where $x<=20000
  return $x
```

(b) Example XQuery

Figure 5.5: Predicate Selectivity Computation

As a concrete example, consider the VSX `/FIXML/Order/OrderQty/@Cash` featured in the example XQuery template of Figure 5.3. For this VSX, predicate selectivities are computed using the XQuery instance shown in Figure 5.5(b).

Plan Selectivity: Similar to the implementation in the relational world, the Plan Selectivity of a VSX is extracted from the annotated plan tree produced by the optimization process. It

is evaluated as the selectivity of the *parent* node of the XML relation associated with the VSX in the plan tree – this parent node is typically a scan operator such as TBSCAN or XISCAN.

5.3.4 Selectivity Estimation Errors

Apart from the above conceptual issue of document versus node selectivities, we have also encountered what appear, within our limited understanding, to be bugs in the DB_XML implementation of selectivity estimation. Specifically, when the constants used in the VSX are *negative* values, then the Predicate and Plan selectivity estimations are way off from what would be expected from the statistical summaries. Secondly, when the VSX attribute is of type *string*, as opposed to integer or float, the selectivity estimations again bear little relationship to the expected values. These situations are explicitly shown later in Section 5.5.5 of the experimental study.

5.4 Plan Parsing

To cater to the complex data processing requirements of XML, the DB_XML engine has incorporated new XML-specific operators, which appear in the execution plan trees. These new operators include XSCAN, XISCAN, XANDOR and XUNNEST, and a sample execution plan featuring some of them is shown in Figure 5.6. All the operators have been taken into account while identifying unique plan trees in the process of production of plan diagrams.

We briefly describe the operators below. For a detailed enumeration of the operators, we refer the reader to [4, 57].

XSCAN: The XSCAN operator is employed to evaluate a single XPath expression. It takes references to XML fragments as input and returns the XML elements that satisfy the given XPath expression. The references to XML fragments that are processed by the XSCAN operator are passed by a nested-loop join operator (NLJOIN), and hence XSCAN always occurs in conjunction with the NLJOIN operator.

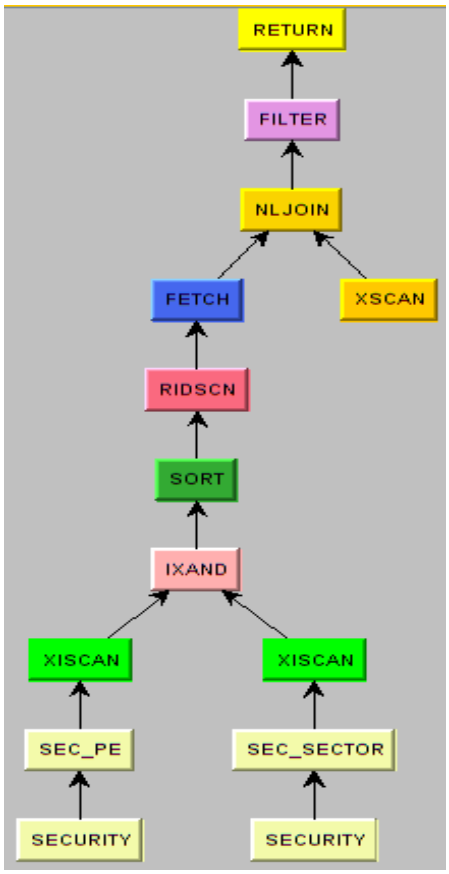


Figure 5.6: Example Plan Tree

XISCAN: This operator is the XML equivalent of a relational index scan. It takes as input, a single query predicate, which in this case is a XPath expression, scans the associated index, and returns the row identifiers of the documents that qualify the search.

XANDOR: The XANDOR (XML index ANDing and ORing) is an n-ary operator that allows evaluation of complex predicates by allowing ANDing and ORing of multiple index scans.

XUNNEST: This operator is used to unnest an XML sequence from an XML data instance. It takes the XML sequence as input and returns XML items along with their sequence numbers.

5.5 Experimental Results

In this section, we describe the experimental framework on which we evaluated the DB_XML optimizer, and the initial results obtained on this framework.

5.5.1 Experimental Setup

All 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. With respect to DB_XML[2], both EXTENDED OPTIMIZATION and HASH-JOIN were enabled. The reason for having chosen a different engine and experimental setup from that of the previous 2 chapters is that DB_XML provided a rich native support for XML data, and the latest version supporting these features were made available to us for the Linux environment. Other popular engines have also provided support for XML data and obtaining optimizer diagrams on these engines would be interesting future work.

5.5.1.1 Databases

We worked with three different XML databases: **TPoX**, **XBench** and **TPCH_X**. TPoX and XBench are native XML benchmarks, while TPCH_X is an XML equivalent of the classical TPCH [62] benchmark used in SQL databases. TPoX models a transaction processing environment, whereas XBench and TPCH_X are representative of decision-support environments – their construction details are given below.

TPoX. The TPoX benchmark database was populated with 50000 CUSTACC, 500000 ORDER and 20833 SECURITY documents, corresponding to the **XXS** scale, which takes about 1GB of space. The data in these documents follow a variety of distributions ranging from uniform to highly skewed, as per the benchmark specifications [59]. The richest set of indexes recommended by the benchmark was created and statistics were collected on all these paths.

XBench. The XBench benchmark [51] supports a choice of four different kinds of databases,

[2]Note for review purposes only: The optimizer is DB2 9.7 pureXML.

namely *TC/SD* (Text centric, Single Document), *TC/MD* (Text centric, Multiple documents), *DC/SD* (Data centric, Single Document) and *DC/MD* (Data centric, Multiple documents). In our study, we chose the **DC/MD** flavor since it appeared to be the most challenging from the optimizer's perspective. Specifically, DB_XML was populated with the "large" option for DC/MD, resulting in a database size of around 1 GB with all data conforming to the uniform distribution. For each of the XQuery templates, indexes were created for all paths appearing in the template and statistics were collected on these paths.

TPCH_X. The NATION, REGION, SUPPLIER and CUSTOMER relations of the TPCH benchmark were converted to their equivalent NATIONS, REGIONS, SUPPLIERS and CUSTOMERS XML_Rs. Further, the ORDER and the LINEITEM relations were combined into the ORDERS XML_R, while the PART and PARTSUPP relations were merged into a single PARTS XML_R. The schemas of the documents stored in these XML_Rs are shown in Figure 5.7. These schemas were constructed and the data was generated using the examples provided with the Toxgene tool [6].

The database was populated with around 1GB of data, and indexes were created for all paths present in the XQuery templates. For each of the 6 tables shown in the schema, multiple documents were loaded, with an entity per document i.e., the NATIONS, REGIONS, CUSTOMERS, SUPPLIERS, PARTS and ORDERS tables were populated with one document per nation, region, customer, supplier, part and order entity, respectively. Each order document in turn included its corresponding lineitems and each part document had its supply information inlined. The number of documents populated in each of these tables is tabulated in Table 5.1 and a sample document for each of the 5 tables is given in Appendix C. The database was made roughly equivalent to the TPCH benchmark in terms of the cardinalities of the tables (except for ORDERS, which was populated with only 300000 documents as opposed to a target value of 1500000 – this was due to memory heap overflow problems occurring while generating data using Toxgene). Also, the generated data was uniformly distributed over the associated domains.

5.5.1.2 Query Templates

We have considered only *two-dimensional* XQuery templates in this study. These templates have been verified to be compatible with the constraints specified for legal XQuery templates

```
create table nations (nations xml)
create table regions (regions xml)
create table customers (customers xml)
create table suppliers (suppliers xml)
create table parts (parts xml)
create table orders (orders xml)
```

Figure 5.7: Database Schema for TPCH_X

Table Name	Number of Documents
nations	25
regions	5
customers	150000
suppliers	10000
parts	200000
orders	300000

Table 5.1: Document Count for TPCH_X

in Section 5.2. All the predicates are on floating-point element values. The plan diagrams are produced at a resolution of 300 points in each dimension, resulting in almost a *hundred thousand* queries being optimized over the space. The DB_XML optimizer itself was run at its default optimization level of 5.

As described in Section 5.3, the constants to be used in the query templates are estimated through a linear-interpolation-based inverse transform of the histogram summaries obtained through the server side utility. Using these constants, we explicitly output a *Selectivity Log* that explicitly tabulates the three types of selectivities delineated in Section 5.3, namely, *Expected*, *Predicate* and *Plan* selectivities. Additionally, the log also indicates the relative and absolute differences between the Expected and Predicate selectivity values.

Also, the cost-increase threshold used for plan diagram reduction in all our experiments is $\lambda = 20\%$. The choice of this setting was based on the observations of [10], wherein anorexic reduction was usually observed at this value for SQL-based query templates.

5.5.2 TPoX

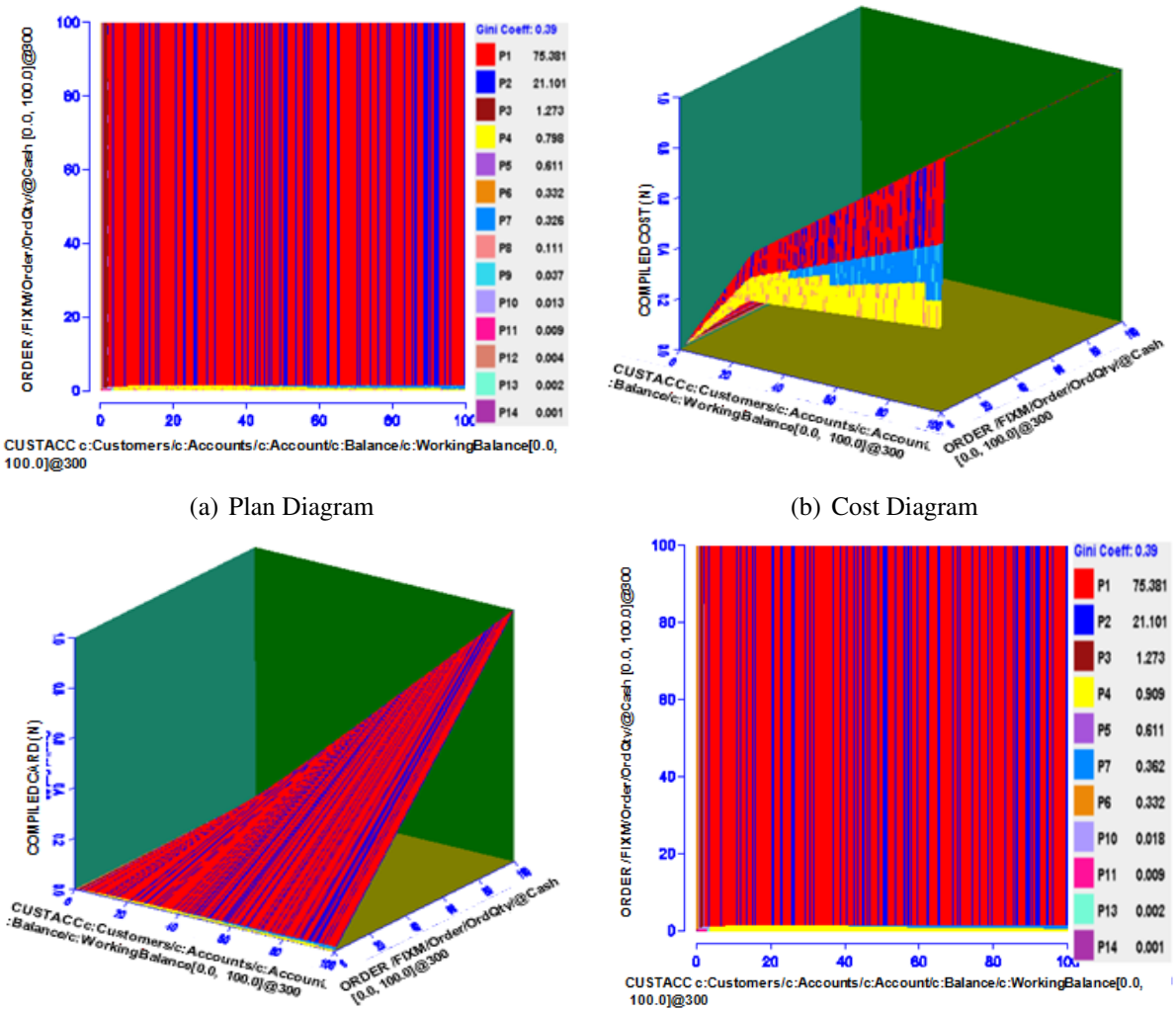
We constructed a XQuery template from the cust_sold_security.xqr XQuery provided with the TPoX benchmark, as shown in Figure 5.8. This template, which we will hereafter refer to as QTX_SEC, returns the customer accounts whose working balance is within a parametrized value and which have been used to trade one or more securities, and, for each of these accounts, the trading amounts of the associated orders being within a parametrized value, the result being alphabetically sorted on Account titles.

```
#NAMESPACE declare default element namespace
  "http://www.fixprotocol.org/FIXML-4-4";
#NAMESPACE declare namespace c="http://tpox-benchmark.com/custacc";
#PSX /c:Customer/c:Accounts/c:Account/c:Balance/c:WorkingBalance
  CUSTACC CUSTACC CADOC
#PSX /FIXML/Order/OrdQty/@Cash ORDER ORDER ODOC
# – #

XQUERY
    declare default element namespace "http://www.fixprotocol.org/FIXML-4-4";
    declare namespace c="http://tpox-benchmark.com/custacc";
        for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")
            /c:Customer/c:Accounts/c:Account[c:Balance/c:WorkingBalance :varies]
        for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order
            [@Acct=$cust/@id/fn:string(.) and OrdQty/@Cash :varies]
        order by $cust/c:AccountTitle/text()
        return
            <Customer>
              {$cust/c:AccountTitle}
              {$cust/c:Currency}
            </Customer>
```

Figure 5.8: XQuery Template for TPoX (QTX_SEC)

Optimizer Diagrams. The optimizer diagram suite for QTX_SEC is shown in Figures 5.9(a) through 5.9(d). The plan diagram consists of 14 plans with plan P1 occupying about three-quarters of the space and plan P14 present in only 0.001% of the diagram, resulting in an overall Gini co-efficient of 0.39. Further, the number of plans decreases to 6 when parameter differences between plan trees are not considered. The low density of plans may perhaps be



(a) Plan Diagram (b) Cost Diagram

(c) Card Diagram (d) Reduced Plan Diagram (λ = 20%)

Figure 5.9: Optimizer Diagrams for TPoX – QTX_SEC (X-Axis: CUSTACC /Customer/Accounts/Account/Balance/WorkingBalance, Y-Axis: ORDER /FIXML/Order/OrdQty/@Cash)

due to the queries being simpler, from an optimization perspective, in a transaction processing environment. 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 horizontal stripe of plan P4, and the brown and orange vertical bands of plans P3 and P6, respectively.

The operator trees for plans P1 (red) and P2 (blue) are shown in Figures 5.10(a) and 5.10(b), respectively. We see here that while they both have the same join order (for ease of discussion,

we have removed annotations from the join order), namely CUSTACC⋈ORDER, there is a structural difference between the two plans due to the re-positioning of an NLJOIN-XSCAN pair. As is evident in the trees, the indexes ACCT_BAL and ORDER_ACCOUNTID are used to scan the CUSTACC and ORDER relations, respectively. The purple vertical stripe (plan P5) also has the same join order as that of plans P1 and P2, but uses the index ORDER_CASH, instead of ORDER_ACCOUNTID, to access the ORDER relation.

In the region near and parallel to the X-axis, the join order changes to ORDER⋈CUSTACC for plans P4, P7, P8, P9 and P14, with all five collectively occupying less than 2% of the area.

Finally, when the plan diagram is reduced with $\lambda = 20\%$, the number of plans comes down to 11, as shown in Figure 5.9(d). The blue vertical bands of plan P2 require a λ of 70% in order to disappear due to swallowing by the red plan P1.

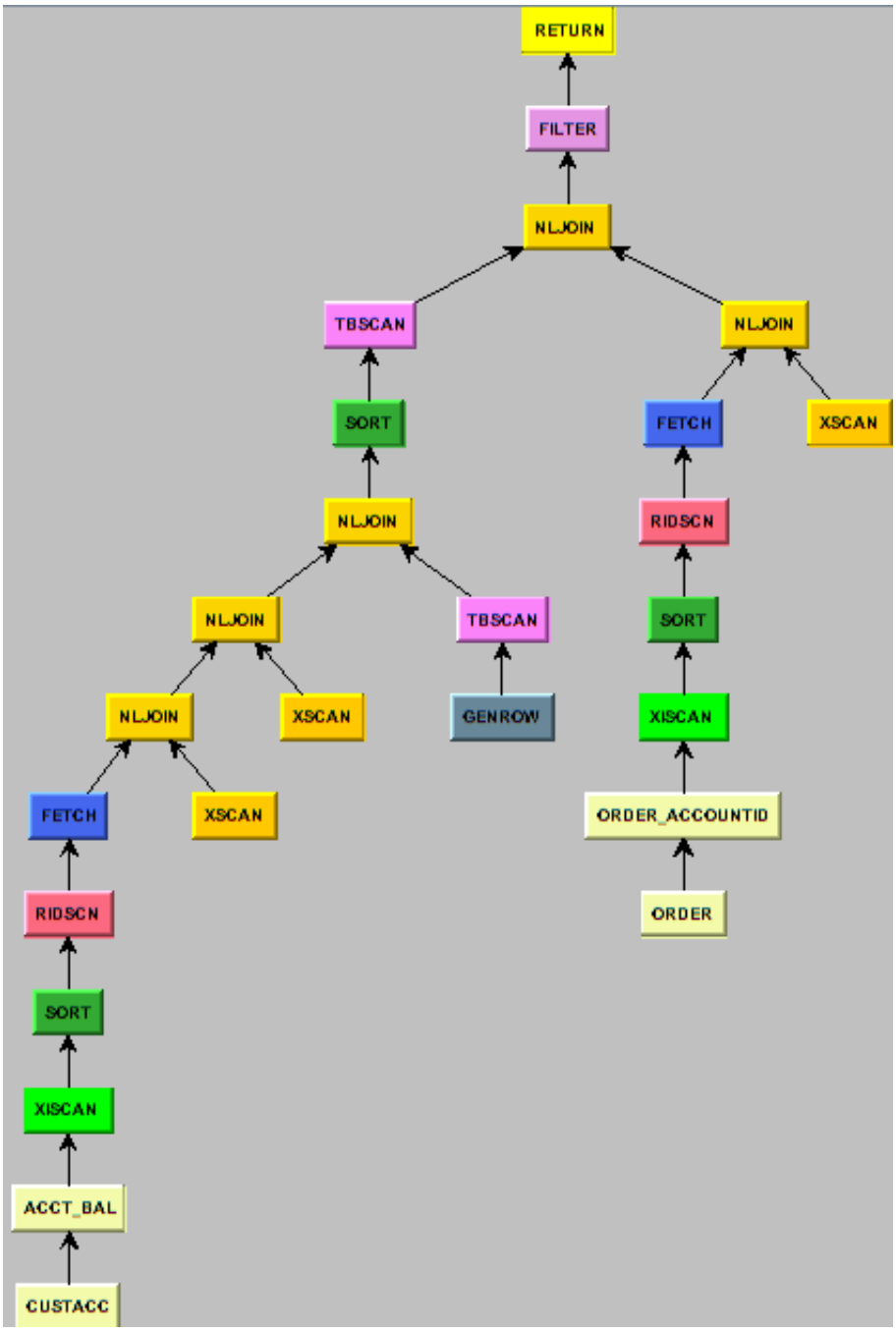
The cost and card diagrams are as shown in Figures 5.9(b) and 5.9(c). Interestingly, we observe here that while the result cardinality has a simple affine relationship with the VSX selectivities, the cost diagram in contrast exhibits a highly non-linear behavior.

Selectivity Logs. The first and last five entries of the selectivity logs for the `/Customer/Accounts/Account/Balance/WorkingBalance` and `/FIXML/Order/OrdQty/@Cash` VSXs are shown in Figures 5.11(a) and 5.11(b), respectively.

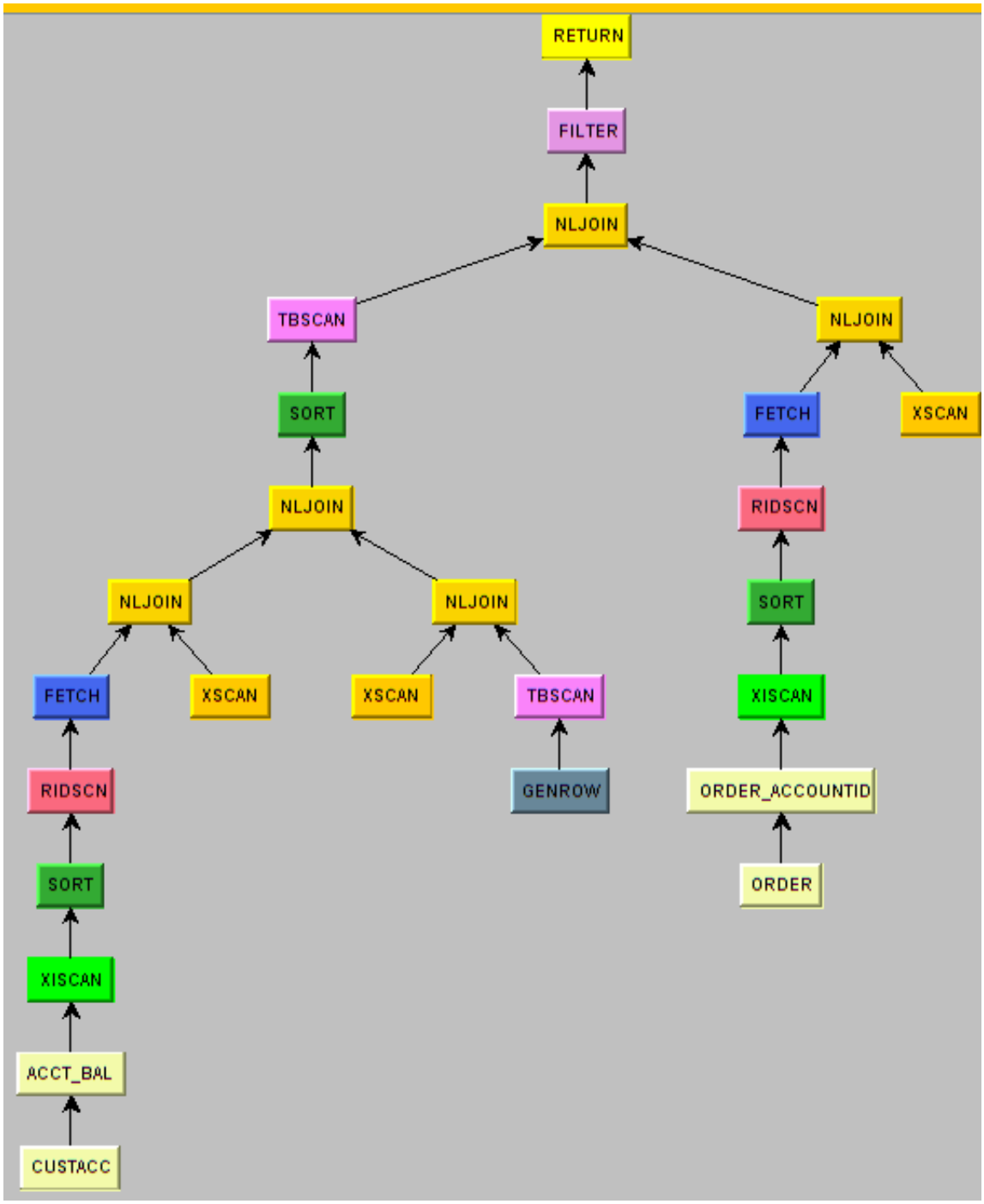
We observe here that the Expected and Predicate selectivities are comparable throughout the selectivity range. However, with regard to Plan selectivities, we observe certain discrepancies in both the VSXs – for the `WorkingBalance` VSX, the selectivities are accurate till the 25.70% mark but subsequently saturate and remain at this level, while with the `@Cash` VSX, they are actually 0% for most of the range.

5.5.3 XBench

We now turn our attention to the XBench native XML benchmark. For this benchmark, a representative XQuery template, QTX19, which is based on XQuery 19 of the queries listed for the DC/MD database, is shown in Figure 5.13. It attempts to retrieve all orders within a parametrized total value for which the associated customers are located within a parametrized


RETURN
FILTER
NLJOIN
TBSCAN
NLJOIN
SORT
FETCH
XSCAN
NLJOIN
RIDSCN
NLJOIN
TBSCAN
SORT
NLJOIN
XSCAN
GENROW
XISCAN
FETCH
XSCAN
ORDER_ACCOUNTID
RIDSCN
ORDER
SORT
XISCAN
ACCT_BAL
CUSTACC

(a) Plan P1 (Red)



(b) Plan P2 (Blue)

Figure 5.10: Plan trees (QTX_SEC)

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	2624.68	0.16	6.25%	0.01	0.16
0.50	5808.76	0.48	4.16%	0.02	0.48
0.83	8758.70	0.78	6.41%	0.05	0.78
1.17	11986.19	1.10	6.36%	0.07	1.10
1.50	15340.54	1.44	4.17%	0.06	1.44
..	..	..	..	..	..
98.50	985164.22	98.52	0.02%	0.02	25.70
98.83	988508.03	98.85	0.02%	0.02	25.70
99.17	991764.72	99.18	0.01%	0.01	25.70
99.50	995074.38	99.51	0.01%	0.01	25.70
99.83	998352.54	99.84	0.01%	0.01	25.70

(a) **VSX: /Customer/Accounts/Account/Balance/WorkingBalance**

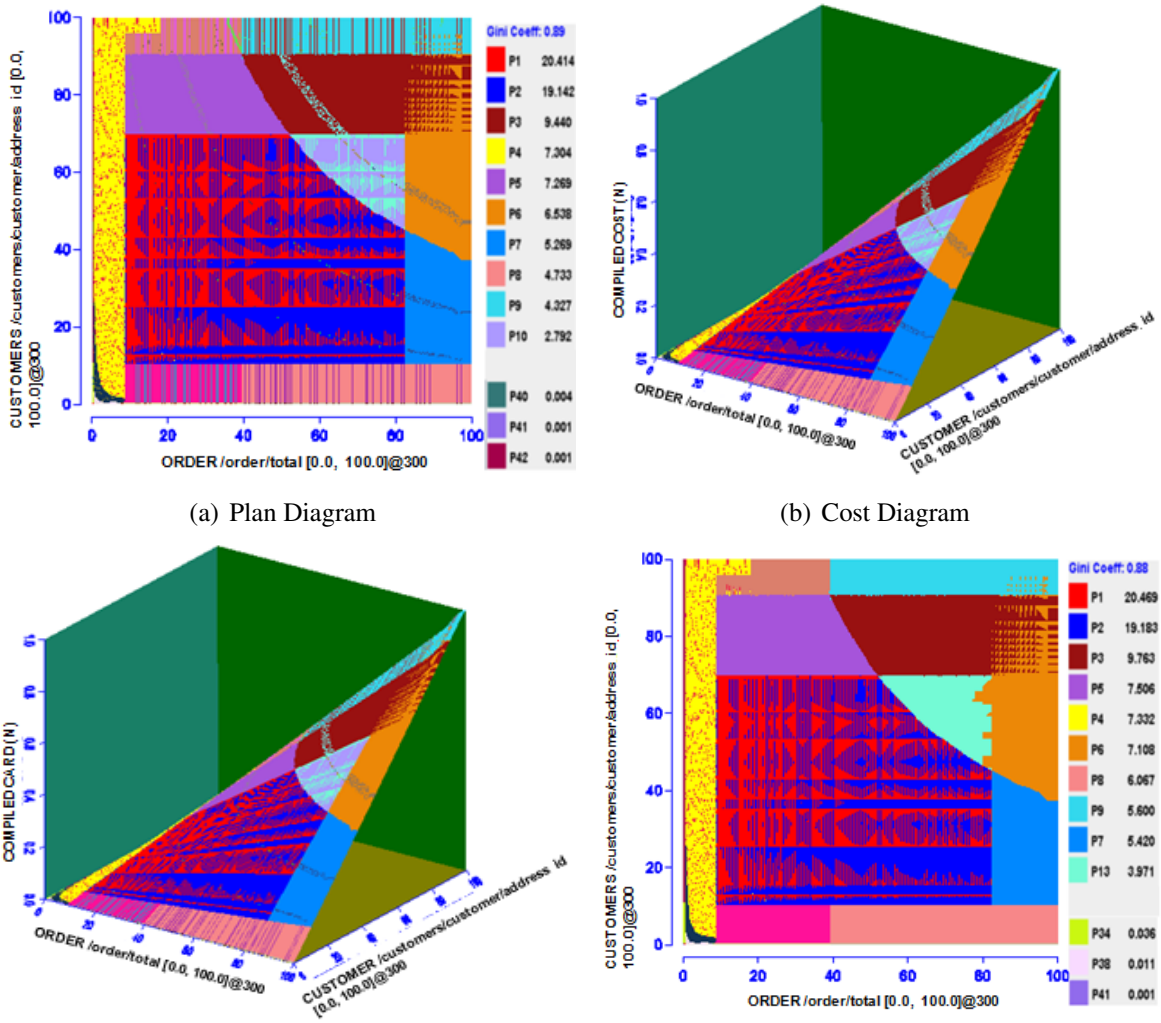
Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	9.44	0.17	0.00%	0.00	0.17
0.50	26.33	0.51	1.96%	0.01	0.51
0.83	43.13	0.84	1.19%	0.01	0.00
1.17	59.34	1.17	0.00%	0.00	0.00
1.50	75.58	1.49	0.67%	0.01	0.00
..	..	..	..	..	..
98.50	4925.19	98.48	0.02%	0.02	0.00
98.83	4941.99	98.82	0.01%	0.01	0.00
99.17	4958.95	99.16	0.01%	0.01	0.00
99.50	4975.97	99.50	0.00%	0.00	0.00
99.83	4992.69	99.83	0.00%	0.00	0.00

(b) **VSX: /FIXML/Order/OrdQty/@Cash**

Figure 5.11: Selectivity Logs (QTX_SEC)

address value, the result being sorted by the order date.

Optimizer Diagrams. The optimizer diagram suite for QTX19 is shown in Figures 5.12(a) through 5.12(d). We see here that the plan diagram consists of 42 plans, and the area distribution of these plans is highly skewed, with the largest plan occupying a little over 20% of the space and the smallest taking 0.001%, the overall Gini co-efficient being as high as 0.89. Most of the differences between the plans are subtle, arising out of their operator parameters, rather than the plan tree structures themselves – for example, a common parameter difference between plans



(a) Plan Diagram (b) Cost Diagram

(c) Card Diagram (d) Reduced Plan Diagram (λ = 20%)

Figure 5.12: Optimizer Diagrams for XBench – QTX19 (X-Axis: ORDER /order/total, Y-Axis: /customers/customer/address_id)

is attributable to the TMPCMPRS parameter, which is present in the TEMP and SORT operators. When such parameter differences are ignored, the number of plans comes down sharply to just 10.

In Figure 5.12(a), plan P2 (dark blue) blends into the plan P1's region (red) in a wave-like pattern. The plan trees for P1 and P2 are shown in Figures 5.14(c) and 5.14(b), respectively. We see here that the plans have different join orders – P1 computes ORDER⋈CUSTOMER whereas P2 evaluates CUSTOMER⋈ORDER – that is, they differ on which relation is outer and which is

```
#PSX /order/total ORDER ORDER ORDER
#PSX /customers/customer/address_id CUSTOMER
  CUSTOMER CUSTOMER
#-#
XQUERY
  for $order in db2-fn:xmlcolumn("ORDER.ORDER")/order,
    $cust in db2-fn:xmlcolumn
      ("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>
```

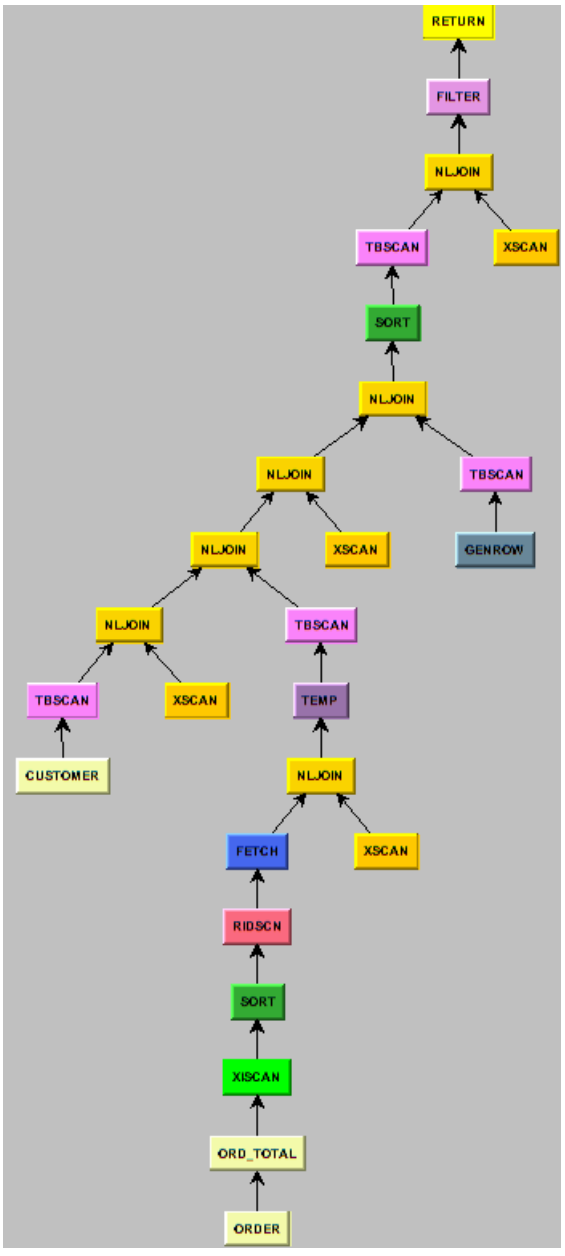
Figure 5.13: XQuery template for XBench (QTX19)

inner in the join. In addition, there are also a few other structural differences between the plans.

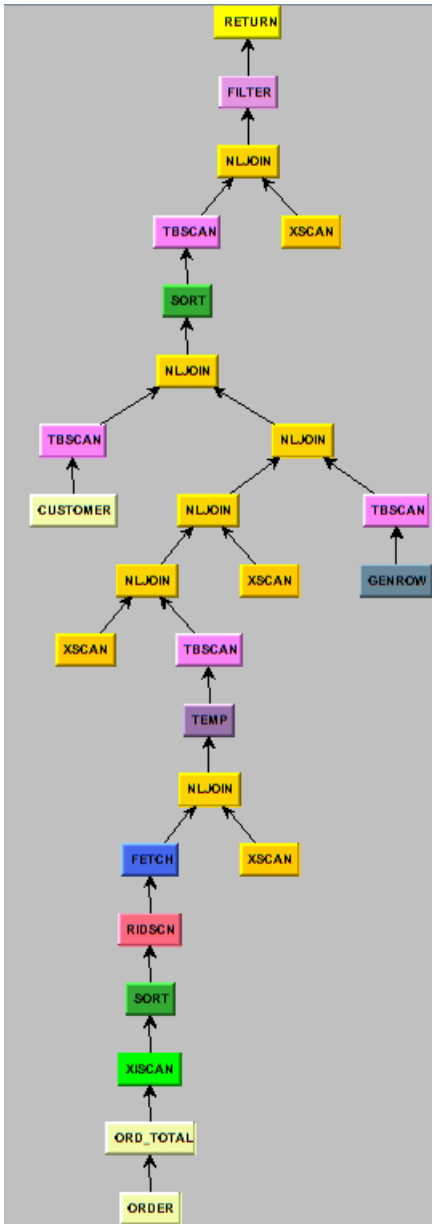
Near and parallel to the Y-axis, we see a yellow vertical strip (plan P4) which is sprinkled with light orange spots (plan P16). Plan P4 is shown in Figure 5.14(a) and differs from P16 only in the positioning of an NLJOIN-XSCAN pair.

The cost and card diagrams are shown in Figures 5.12(b) and 5.12(c), respectively. We observe here a simple affine relationship for both cost and cardinality with regard to the selectivity values.

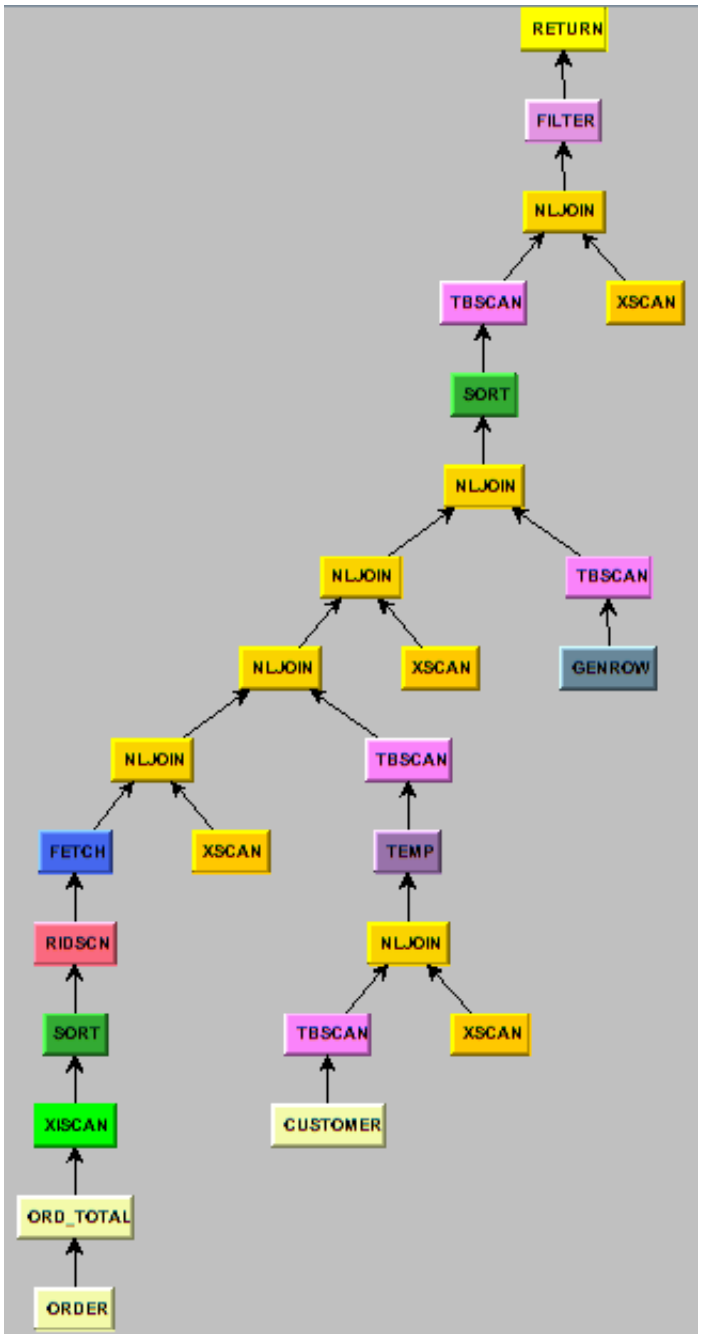
When reduction with $\lambda = 20\%$ is applied to the plan diagram, the number of plans goes down from 42 to 21, as shown in Figure 5.12(d). Interestingly, the wave pattern produced by plan P2 remains intact – it is eliminated only when λ is increased to 30%. However, it is possible that the P2-associated wave pattern would have been removed at a lower λ if our reduction algorithm had supported *partial* swallowing, wherein a *subset* of the points associated with a plan are replaced – currently, the reduction scheme used only supports total swallowing, allowing a plan's points to be replaced only if *all* its points are replaced.



(a) Plan P4 (Yellow)



(b) Plan P2 (Blue)



(c) Plan P1 (Red)

Figure 5.14: Plan Trees (QTX19)

Selectivity Logs. The first and last five entries of the selectivity logs corresponding to the `/order/total` and `/customers/customer/address_id` VSXs are shown in Figures 5.15(a) and 5.15(b), respectively. The Expected selectivities for `/order/total` show marginal differences with regard to the Predicate selectivities for the initial entries and, with increasing selectivity, eventually become close to these values. However, Plan selectivity values remain close to the expected values only until the 50% mark, after which they saturate and remain the same irrespective of the constant used, similar to our observations in the TPoX environment.

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	53.78	0.21	19.05%	0.04	0.21
0.50	117.30	0.74	32.43%	0.24	0.74
0.83	180.82	1.27	34.65%	0.44	1.27
1.17	244.35	1.80	35.00%	0.63	1.80
1.50	307.87	2.33	35.62%	0.83	2.33
..	..	..	..	..	..
98.50	11730.58	97.49	1.03%	1.01	50.00
98.83	11822.00	98.25	0.59%	0.58	50.00
99.17	11913.43	99.01	0.16%	0.16	50.00
99.50	12004.86	99.77	0.27%	0.27	50.00
99.83	12096.28	100.00	0.17%	0.17	50.00

(a) **VSX: /order/total**

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	962.47	0.17	0.00%	0.00	0.00
0.50	2887.42	0.50	0.00%	0.00	0.00
0.83	4812.37	0.84	1.19%	0.01	0.00
1.17	6737.33	1.17	0.00%	0.00	0.00
1.50	8662.28	1.50	0.00%	0.00	0.00
..	..	..	..	..	..
98.50	567229.38	98.48	0.02%	0.02	0.00
98.83	569177.96	98.60	0.23%	0.23	0.00
99.17	571126.54	98.60	0.58%	0.57	0.00
99.50	573075.12	98.60	0.91%	0.90	0.00
99.83	575023.70	98.60	1.25%	1.23	0.00

(b) **VSX: /customers/customer/address_id**

Figure 5.15: Selectivity Logs (QTX19)

On the other hand, for /customers/customer/address_id, the Expected selectivities match the Predicate selectivities with tolerable deviations for most of the selectivity range – however, the Predicate selectivities for the last four entries (98.83% through 99.83%) remain static at a value of 98.60%. This is in spite of close to 3000 records being present between the constants associated with 98.83% and 99.83%. Further, our plan selectivity extraction mechanism does not work even partially in this case, returning a value of 0% throughout the selectivity range.

5.5.4 TPCH_X

In our earlier work on SQL databases [52], we had constructed a variety of TPCH-based parametrized SQL query templates. For example, QT5, the template shown in Figure 5.16(a), is based on TPCH Query 5. This query lists the revenue volume done through local suppliers, when the c_acctbal of the CUSTOMER relation and the s_acctbal of the SUPPLIER relation are within their respective parametrized values. XQuery templates similar to these SQL templates were created for evaluating DB_XML – as a case in point, the QTX5 template shown in Figure 5.16(b) is similar to QT5.

Optimizer Diagrams. The suite of optimizer diagrams for QTX5 is shown in Figures 5.17(a) through 5.17(d). Further, for comparative purposes, we also show the plan diagram for the SQL version, QT5, in Figure 5.17(e).

In Figure 5.17(a), we can see that there are 25 plans in the plan diagram with a wide variation in the areas covered by these plans. The top three plans occupy more than 70% of the space, and the last three occupy only 0.068% of the region – this skew in area coverage is captured through the Gini coefficient, whose value is 0.78. When parameter differences between plan trees are not considered, the number of plans reduces to 16 in the plan diagram.

The plan diagram is also characterized by a variety of intricate patterns, and has several regions where many plans intermingle – the most prominent example is found in the upper half region of the plan diagram where vertical bands of Plan P1 (red), Plan P2 (blue), Plan P3 (brown) and Plan P6 (orange) alternate with one another in a convoluted manner. Yet another example is the intermingling of plan P1 (red), plan P3 (brown), plan P5 (purple) and plan P9

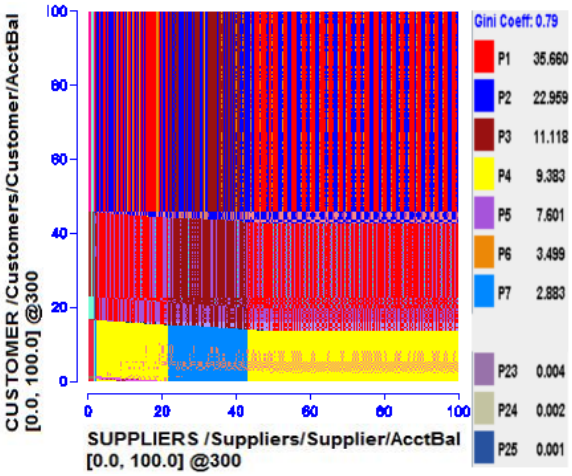
```
select
  n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
from
  customer, orders, lineitem, supplier, nation, region
where
  c_custkey = o_custkey and l_orderkey = o_orderkey and
  l_suppkey = s_suppkey and c_nationkey = s_nationkey and
  s_nationkey = n_nationkey and n_regionkey = r_regionkey
  and r_name = 'ASIA' and o_orderdate >= '1994-01-01'
  and o_orderdate < '1995-01-01' and
  c_acctbal :varies and s_acctbal :varies
group by n_name
order by revenue desc
```

(a) **SQL template for TPCH (QT5)**

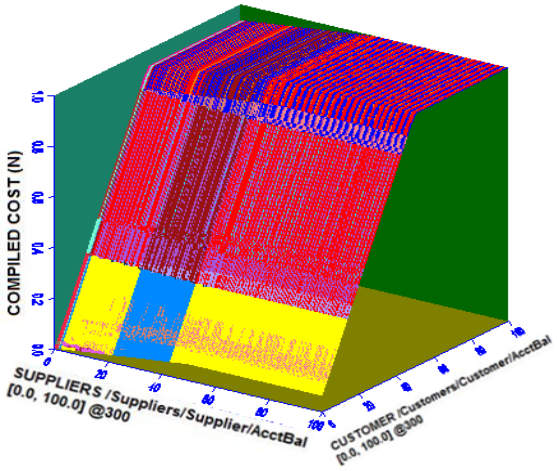
```
#VSX /Suppliers/Supplier/AcctBal SUPPLIERS SUPPLIERS SUPPLIERS
#VSX /Customers/Customer/AcctBal CUSTOMERS
  CUSTOMERS CUSTOMERS
# – #
XQUERY for $reg in db2-fn:xmlcolumn("REGIONS.REGIONS")
  /Regions/Region[Name=" AFRICA"]
  for $nat in db2-fn:xmlcolumn("NATIONS.NATIONS")
    /Nations/Nation[xs:double(RegionKey)=$reg/@key/xs:double(.)]
    let $sup := db2-fn:xmlcolumn("SUPPLIERS.SUPPLIERS")/Suppliers
        /Supplier[(xs:double(NationKey)=$nat/@key/xs:double(.))
          and (AcctBal :varies)],
    $cust := db2-fn:xmlcolumn("CUSTOMERS.CUSTOMERS")/Customers
      /Customer[(xs:double(NationKey)=$sup/NationKey/xs:double(.))
        and (AcctBal :varies)],
    $line := db2-fn:xmlcolumn("ORDERS.ORDERS")/Orders/Order
      [xs:double(CustKey) = $cust/@key/xs:double(.)
        and OrderDate[ . > "1994-01-01" and . < "1997-01-01"]]
          /LineItem[xs:double(SuppKey) = $sup/@Key/xs:double(.)]
    let $summand :=(for $x in $line
        return $x/ExtendedPrice*(1 -$x/Discount))
    order by sum($summand)
    return
        <result> {$nat/@key} {$nat/Name}
          <revenue> {sum($summand)} </revenue>
        </result>
```

(b) **XQuery template for TPCH_X (QTX5)**

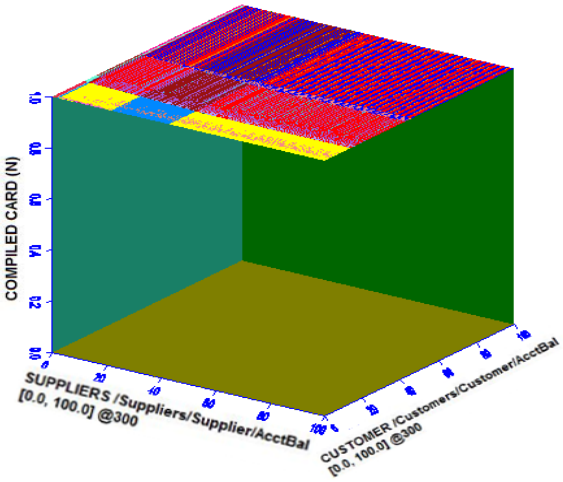
Figure 5.16: Query Templates



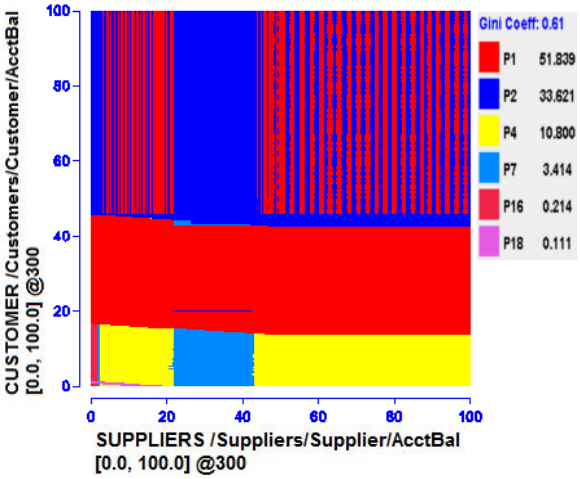
(a) Plan Diagram



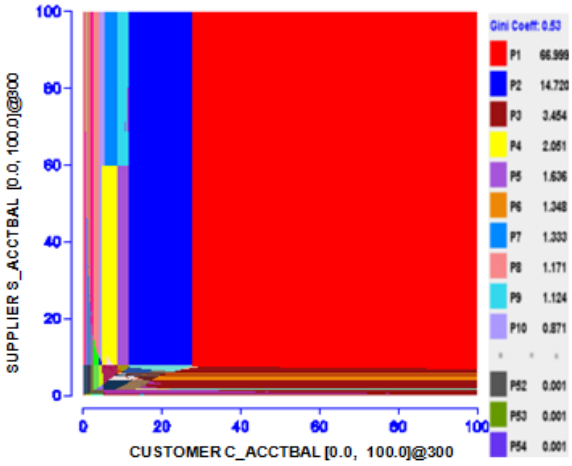
(b) Cost Diagram



(c) Card Diagram



(d) Reduced Plan Diagram (λ = 20%)



(e) Plan Diagram (QT5)

Figure 5.17: Optimizer Diagrams for TPCH_X– QTX5 (X-Axis: SUPPLIERS /Suppliers/Supplier/AcctBal, Y-Axis: CUSTOMERS /Customers/Customer/AcctBal

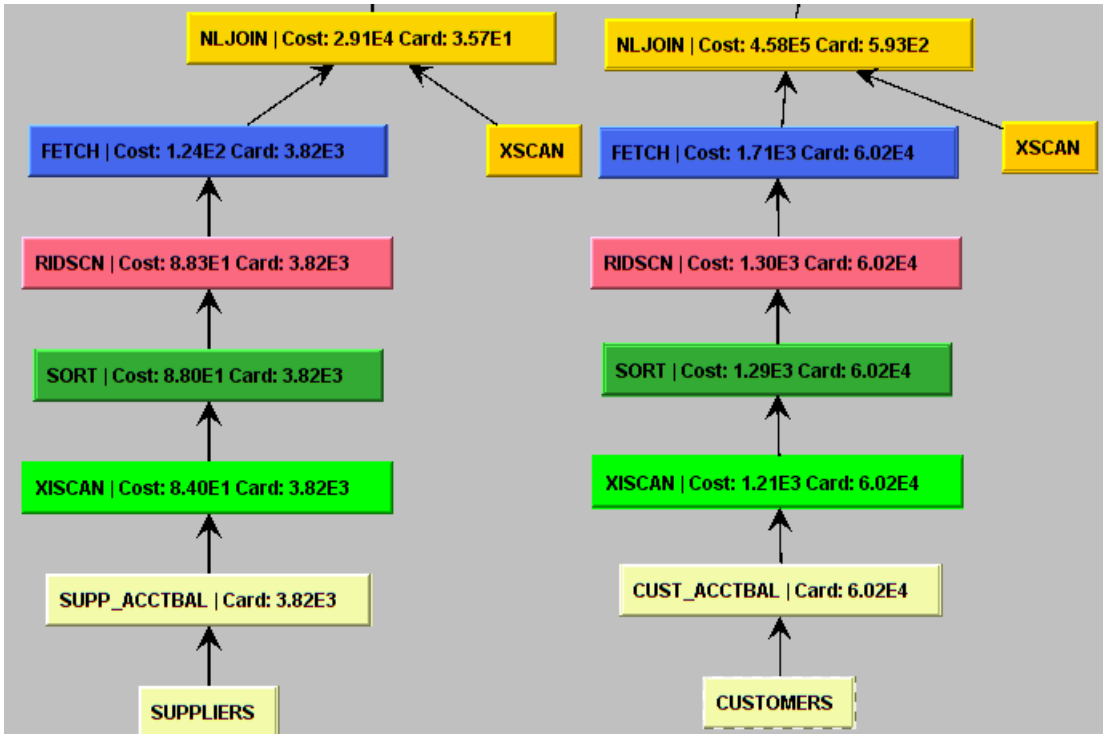
(turquoise) in the lower half of the diagram.

Let us denote the XML_Rs in the XQuery template of Figure 5.16(b), namely REGIONS, NATIONS, SUPPLIERS, CUSTOMERS and ORDERS, as R, N, S, C and O, respectively. When we drill down into the plan internals, we find that there are only three different join orders in the whole plan space! Plans P1, P3, P13 and P17 have the left-deep join order (((R⋈N)⋈S)⋈C)⋈O, plans P2, P5, P6, P9, P10, P11, P14, P15 and P20 have the bushy join order ((R⋈N)⋈(S⋈C))⋈O and plans P4, P7, P8, P12, P16, P18, P19 and P21-P25 also have a bushy join order which is ((R⋈N)⋈S)⋈(C⋈O). The explicit trees for these plans are too large to include here, and have therefore been made available at our project URL [53].

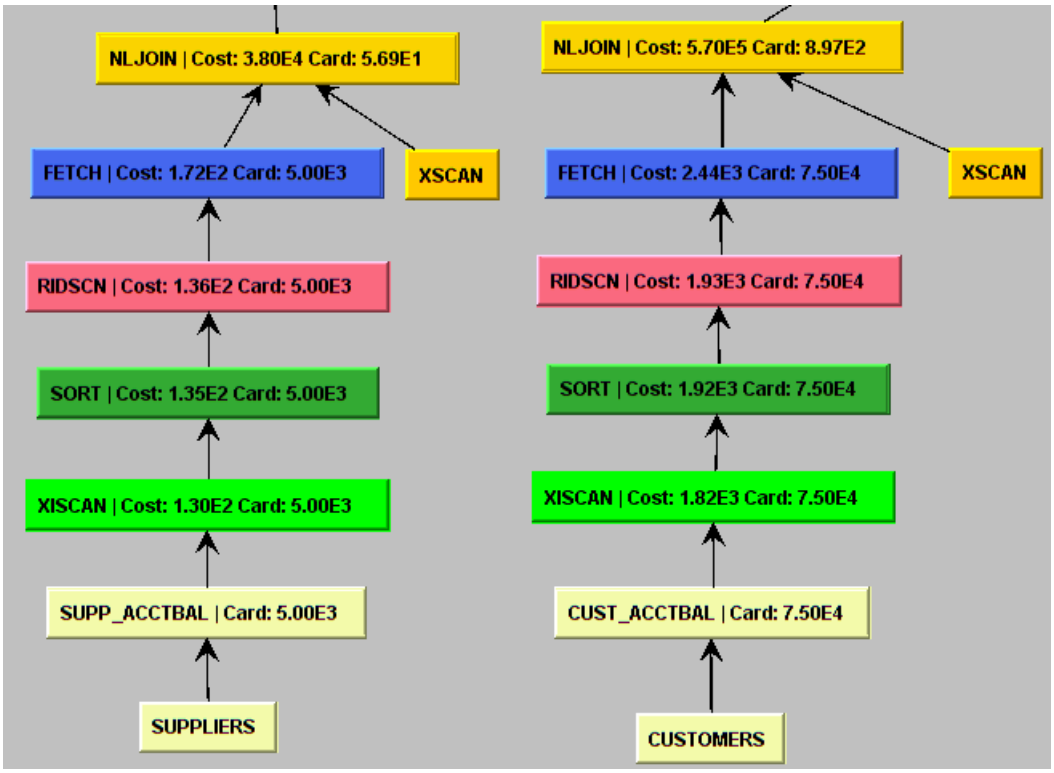
Turning our attention to the Cost Diagram (Figure 5.17(b)), we observe that the cost steadily increases with the increase in selectivity of the CUSTOMERS XML_R. The effect of the SUPPLIERS relation seems mild in comparison, perhaps because the CUSTOMERS relation is about 15 times the size of the SUPPLIERS relation. We also see that the cost plateaus out when approximately 50% selectivity is reached in the VSX dimensions. To analyze this behavior, we looked into the annotated plan trees at different points in the selectivity space. Annotated plan trees at roughly 40%, 60% and 98% selectivities in both dimensions are partially shown in Figures 5.18(a), 5.18(b) and 5.18(c). The total cost of these plans being 4.87e5, 6.08e5 and 6.10e5, respectively, it can be inferred by looking at these subtrees that the overall costs are dominated by the NLJOIN operator in the CUSTOMERS subtree. For a selectivity increase selectivity from 40% to 60%, a cost increase of 1.12e5 is observed in the NLJOIN operator corresponding to the CUSTOMERS relation. However, on moving from 60% to 98%, the cost goes up by merely 0.01e5. It appears puzzling that the NLJOIN cost does not scale up at least linearly with increasing selectivity and this issue needs further investigation.

Moving on to the cardinality diagram (Figure 5.17(c)), it shows a constant result set size of 1 throughout the space since the query computes an aggregate operation (*sum*) in the final step.

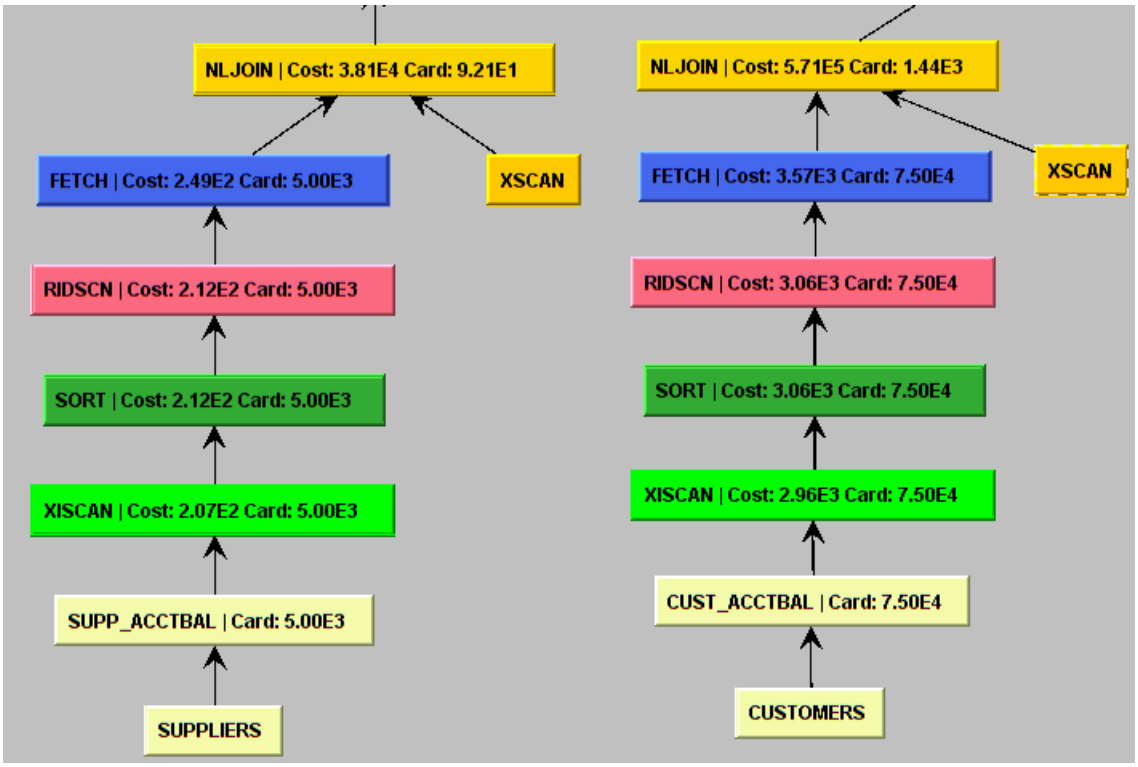
Now, when the plan diagram is subject to $\lambda = 20\%$ reduction, the original 25 plans come down to 6, as shown in Figure 5.17(d). Further, several of the intricate patterns are eliminated – for example, a part of the lower half region of rapidly alternating plans is singly dissolved by the red plan (Plan P1).



(a) 40% Selectivity



(b) 60% Selectivity



(c) 98% Selectivity

Figure 5.18: Annotated Plan trees – Partial (QTX5)

Finally, when we compare the XML and SQL-based plan diagrams in Figures 5.17(a) and 5.17(e), respectively, we notice that the XML diagram is significantly more complex in comparison in terms of their spatial layouts even though the numbers of plans is the SQL-based diagram is more than twice the number found in the XML diagram. This certainly merits further investigation.

Selectivity Logs. The selectivity logs for the `/Suppliers/Supplier/AcctBal` and `/Customers/Customer/AcctBal` VSXs are shown in Figures 5.19(a) and 5.19(b), respectively – specifically, the first five and last five entries are extracted and shown here. The Expected selectivities for `/Suppliers/Supplier/AcctBal` show tolerable differences with regard to the Predicate selectivities for the initial entries and, with increasing selectivity, eventually become close to these values. On the other hand, for `/Customers/Customer/AcctBal`, the Expected selectivities match the Predicate selec-

tivities with tolerable deviations for most of the selectivity range – however, for the last eight Expected selectivity entries (97.50% through 99.83%), the Predicate selectivity stays put at 97.24%. This is in spite of the fact that there are more than 3500 records between the constants associated with 97.50% and 99.83%. Again, this is an issue that needs to be investigated further.

Finally, the Plan selectivity values for both the VSXs show behavior similar to the previous cases – the selectivity values saturate at the 50% mark.

5.5.4.1 Query Complexity Problem

Apart from the above issue with plan selectivities, we also ran into another problem when we evaluated the XQuery template QTX2 shown in Figure 5.20, which is based on Query 2 of TPCH. For this template, the plan diagram features only two very similar plans, the difference being a slight variation in their join orders, with the first NLJOIN operator, which combines the SUPPLIERS and PARTS relations, having SUPPLIERS as the outer relation in P1 (red), and PARTS as the outer relation in P2 (blue). It is certainly in the realm of possibility that this plan diagram genuinely features only a few plans, but we also think that the message issued by the DB_XML optimizer, warning us that the performance of the query might be suboptimal, may have impacted the observed behavior. This warning persisted even when the optimization level was lowered and the heap space was sufficiently increased (upto 3GB). However, when the template was simplified by removing some constructs, the message went away and the resulting diagram in fact featured quite a few more plans than those seen with QTX2.

5.5.5 Selectivity Computation Problems

As pointed out earlier, selectivity estimation in DB_XML seemed to be aberrant when the constants used in the predicate were negative numbers or of the string data type. An example for both of them is presented in this section.

5.5.5.1 Case: Negative Numbers

An extract of the Selectivity log in the case of negative numbers is given in Figure 5.21(a). The XQuery template for which the selectivity log was obtained is shown in Figure 5.22(a) and this

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	19.70	0.19	10.52%	0.02	0.19
0.50	56.50	0.52	3.84%	0.02	0.52
0.83	93.29	0.86	3.48%	0.03	0.86
1.17	130.09	1.19	1.68%	0.02	1.19
1.50	166.88	1.53	1.96%	0.03	1.53
..	..	..	..	..	..
98.50	10842.98	98.57	0.07%	0.07	50.00
98.83	10871.64	98.83	0.00%	0.00	50.00
99.17	10905.05	99.13	0.04%	0.04	50.00
99.50	10947.84	99.52	0.02%	0.02	50.00
99.83	10983.63	99.85	0.01%	0.02	50.00

(a) **VSX: /Suppliers/Supplier/AcctBal**

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	17.43	0.16	6.25%	0.01	0.16
0.50	52.98	0.48	4.16%	0.02	0.48
0.83	90.02	0.82	1.21%	0.01	0.82
1.17	126.47	1.15	0.86%	0.02	1.15
1.50	163.59	1.49	0.67%	0.01	1.49
..	..	..	..	..	..
98.50	10836.41	97.24	1.29%	1.26	50.00
98.83	10871.71	97.24	1.63%	1.59	50.00
99.17	10907.94	97.24	1.98%	1.93	50.00
99.50	10945.37	97.24	2.32%	2.26	50.00
99.83	10981.93	97.24	2.66%	2.59	50.00

(b) **VSX: /Customers/Customer/AcctBal**

Figure 5.19: Selectivity Logs (QTX5)

```
#VSX /Parts/Part/RetailPrice PARTS PART1 PARTS
#VSX /Parts/Part/PartSupp/SupplyCost PARTS PART2 PARTS
#–#
XQUERY
for $part in db2-fn:xmlcolumn("PARTS.PARTS")/Parts/Part
 [RetailPrice :varies]/PartSupp,
  $sup in db2-fn:xmlcolumn("SUPPLIERS.SUPPLIERS")/Suppliers/Supplier,
  $nat in db2-fn:xmlcolumn("NATIONS.NATIONS")/Nations/Nation,
  $reg in db2-fn:xmlcolumn("REGIONS.REGIONS")/Regions/Region
    where xs:double($sup/@Key)=xs:double($part/Suppkey)
      and $sup/NationKey/xs:double(.)=$nat/@key/xs:double(.)
      and $nat/RegionKey/xs:double(.)=$reg/@key/xs:double(.)
      and $reg/Name =" EUROPE"
      and $part/SupplyCost <=(let $reg := db2-fn:xmlcolumn("REGIONS.REGIONS")
        /Regions/Region[Name=" EUROPE"],
        $nat := db2-fn:xmlcolumn("NATIONS.NATIONS")
          /Nations/Nation[xs:double(RegionKey)=$reg/@key/xs:double(.)],
        $sup := db2-fn:xmlcolumn("SUPPLIERS.SUPPLIERS")
          /Suppliers/Supplier[$nat/@key/xs:double(.) = xs:double(NationKey)],
        $partsupp := db2-fn:xmlcolumn("PARTS.PARTS")
          /Parts/Part[xs:double(PartKey)=$part/../PartKey/xs:double(.)]/PartSupp[
          ($sup/@Key/xs:double(.)=xs:double(Suppkey)) and (SupplyCost :varies) ]
              return min($partsupp/SupplyCost))
    order by $sup/AcctBal descending, $nat/Name, $sup/Name, $part/../PartKey
    return <final>
              {$sup/AcctBal}
              {$sup/Name}
              {$nat/Name}
              {$part/../PartKey}
              {$part/../Mfgr}
              {$part/SupplyCost}
              {$sup/Address}
              {$sup/Phone}
              {$sup/Comment}
            </final>
```

Figure 5.20: XQuery template for TPCH_X (QTX2)

template corresponds to the TPCH_X database environment. It can be seen that in the Predicate Selectivity column, a constant of 0.26 appears in the first 5 rows. In fact this appears till the numbers in the Constant column of the Selectivity log are in the domain of negative values, and, changes to correct estimation values when the positive domain is entered. This seems to

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	-968.17	0.26	34.61%	0.09	0.50
0.50	-941.68	0.26	92.30%	0.24	0.50
0.83	-906.37	0.26	219.23%	0.57	0.50
1.17	-871.05	0.26	350.00%	0.91	0.50
1.50	-835.74	0.26	476.92%	1.24	0.50
..	..	..	..	..	..
98.50	9836.67	98.89	0.39%	0.39	0.50
98.83	9872.95	99.22	0.39%	0.39	0.50
99.17	9909.24	99.55	0.38%	0.38	0.50
99.50	9945.52	100.00	0.50%	0.50	0.50
99.83	9981.80	100.00	0.17%	0.17	0.50

(a) **Selectivity Log: Negative Numbers**

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Selectivity
0.17	18.55	0.17	0.00%	0.00	0.17
0.50	55.21	0.50	0.00%	0.00	0.50
0.83	91.87	0.84	1.00%	0.01	0.84
1.17	128.53	1.17	0.00%	0.00	1.17
1.50	165.19	1.50	0.00%	0.00	1.50
..	..	..	..	..	..
98.50	10832.62	98.48	0.02%	0.02	50.00
98.83	10866.25	98.78	0.05%	0.05	50.00
99.17	10904.91	99.14	0.03%	0.03	50.00
99.50	10942.93	99.48	0.02%	0.02	50.00
99.83	10980.96	99.83	0.00%	0.00	50.00

(b) **Selectivity Log: Positive Values**

Expected Selectivity	Constant	Predicate Selectivity	Relative Difference	Absolute Difference	Plan Sel.
0.17	“AAKSOCX”	0.01	1600.00%	0.16	0.01
0.50	“ABGOBJVG”	0.03	1566.66%	0.47	0.03
0.83	“ABXSKUN”	0.03	2666.66%	0.80	0.03
1.17	“ACGYEIN”	0.04	2825.00%	1.13	0.04
1.50	“ACVFJ”	0.04	3650.00%	1.46	0.04
..	..	..	..	..	..
98.50	“WMQFUMK”	87.90	12.05%	10.60	0.00
98.83	“WPOUFSO”	87.94	12.38%	10.89	0.00
99.17	“WTDAICW”	88.00	12.69%	11.17	0.00
99.50	“XMATHH”	91.11	9.20%	8.39	0.00
99.83	“ZAUTV”	91.11	9.57%	8.72	0.00

(c) **Selectivity Log: String Data Type**

Figure 5.21: Selectivity Log Errors

```
XQUERY
 for $x in db2-fn:xmlcolumn("CUSTOMERS.CUSTOMERS")
    /Customers/Customer/AcctBal
 where $x :varies
 return $x
```

(a) **XQuery template: Negative Numbers**

```
XQUERY
 declare default element namespace
  "http://www.fixprotocol.org/FIXML-4-4";
 declare namespace
  c="http://tpox-benchmark.com/custacc";

 for $x in db2-fn:xmlcolumn("ORDER.ODOC")
    /FIXML/Order/Instrmt/@Sym
 where $x :varies
 return $x
```

(b) **XQuery template: String Data Type**

Figure 5.22: XQuery Templates for Selectivity error

be an error since there are quite a few elements in the negative domain. This fact is confirmed by the observation that the errors in Predicate Selectivity estimation disappeared when the data values were linearly translated without affecting their distribution (a constant was added to all the data values, so that all values reside only in the positive domain). The Selectivity log for the linearly translated data is shown in Figure 5.21(b). Data sets corresponding to the case of negative numbers and to the linearly translated version can be downloaded from [53]. In fact, in a test scenario, where *all* the values of a certain VSX were situated in the negative domain, erroneous selectivity estimation was observed *throughout* the range. As a case in point, 100% selectivity was erroneously reported as 0.0%.

5.5.5.2 Case: String Data Type

An extract of the Selectivity log for this case and its corresponding XQuery template are shown in Figures 5.21(c) and 5.22(b), respectively. This particular scenario was stumbled upon in the TPoX database environment. In this case, the Predicate selectivity estimation values, as seen

Benchmark	No of plans	No. of join-orders
TPoX	14	2
XBench	42	2
TPCH_X	25	3

Table 5.2: Join Order Statistics for XML plan diagrams

in the Predicate Selectivity column in Figure 5.21(c) were explicitly verified with the statistical summaries obtained through a server side utility and were found to be inconsistent.

5.6 Structure-based Reduction of XML Plan Diagrams

We have discussed in detail the various features of the XML plan diagrams, including the join orders present, in the previous sections. Here, we present in brief, the results obtained when structure-based reduction techniques of Chapter 3 are applied to these XML plans diagrams

Table 5.2 lists the results obtained for all three benchmarks which were used for the study. We can see that in the case TPoX and XBench, where the QTs have only 2 relations, both possible join orders are present in the plan diagram. It is surprising is that there are 42 plans with just 2 relations. In the case of TPCH_X, only 3 different join orders are present with the plan cardinality being 25. This join order cardinality is rather low in comparison to its SQL counterpart, which had 18 join orders among 54 plans in the plan diagram.

5.7 Semantic Coloring of XML Plan Diagrams

We present in this section semantically colored XML plan diagrams using MDS techniques elaborated in the previous chapter.

A plot of the frequency vs Jaccard distances for all these three benchmarks is given in Figure 5.23. Figures 5.24(a), 5.24(b) and 5.24(c) are the MDS colored plan diagrams for the TPoX, XBench and TPCH_X benchmarks' XQuery templates given in Figures 5.8, 5.13 and 5.16(b), respectively. Finally, the statistics regarding the quality of coloring obtained in all three bench-

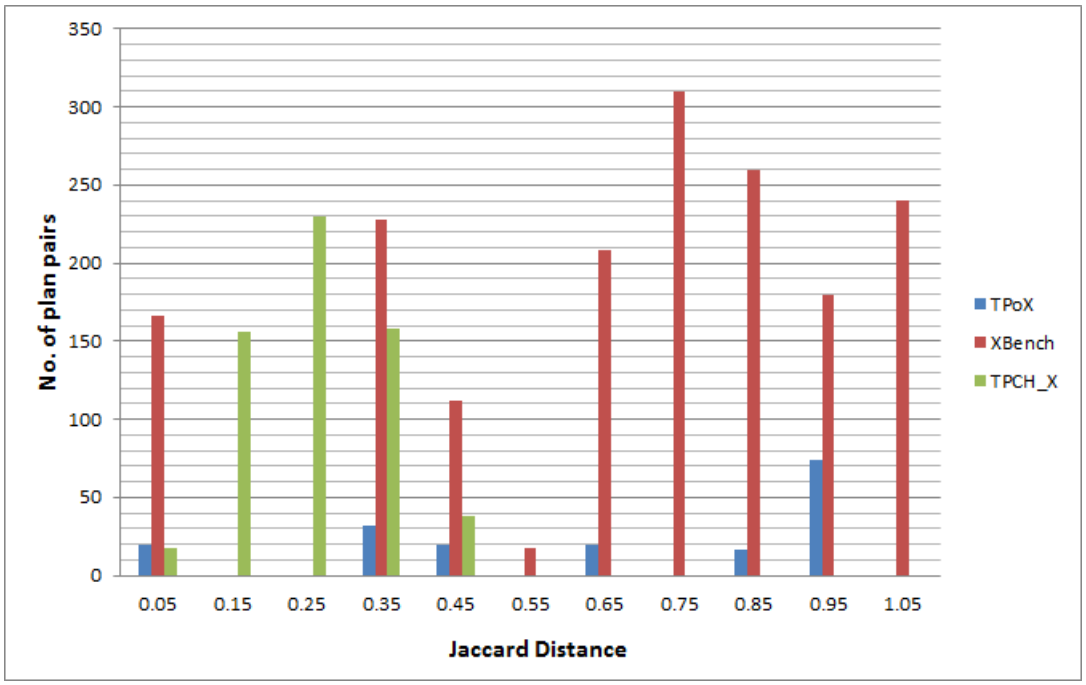


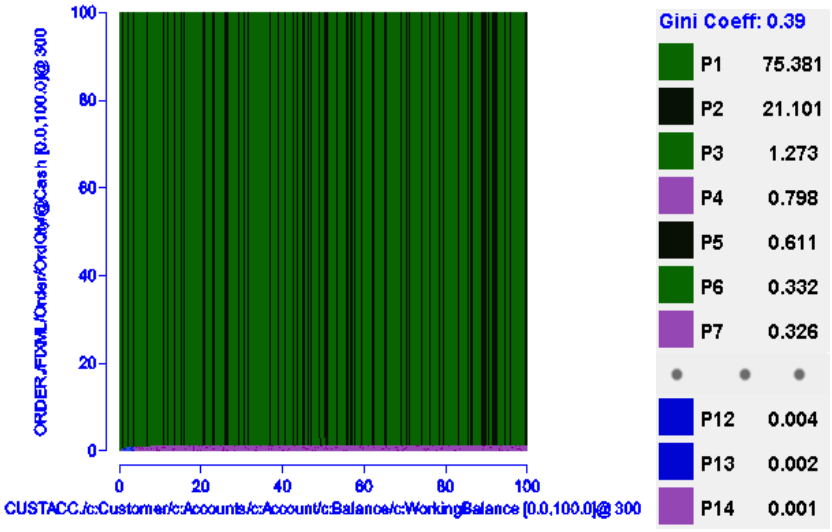
Figure 5.23: Jaccard Distribution in XML Plan Diagrams

marks is tabulated in Table 5.3.

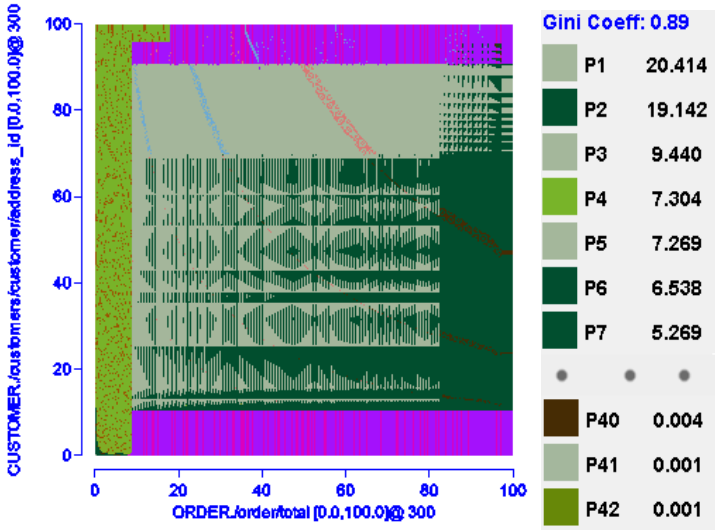
If we look at the distribution of Jaccard distances in Figure 5.23, we can see that the plot for the XBench benchmark has highly conspicuous peaks across its dissimilarity range. The XBench benchmark has the highest number of plans, and the highest maximum Jaccard distance (1.066) among the three benchmarks. TPCH_X, on the other hand has the lowest maximum Jaccard distance of 0.48. The plots for TPoX and XBench have breaks in between, like in the range of 0.15-0.25, with no pair of plans having dissimilarities in this range.

The coloring of the plan diagram of TPCH_X reflects its short dissimilarity range, with similar colors throughout the diagram. The biggest color distance is between plans P14 and P21 (0.53), which also have the maximum Jaccard dissimilarity between them (0.48). The location of these plans are hard to point out since they occupy very less area. In the case of TPoX, it is easy to observe that more than 96% of the plan diagram is occupied by the top two plans which are colored green and black.

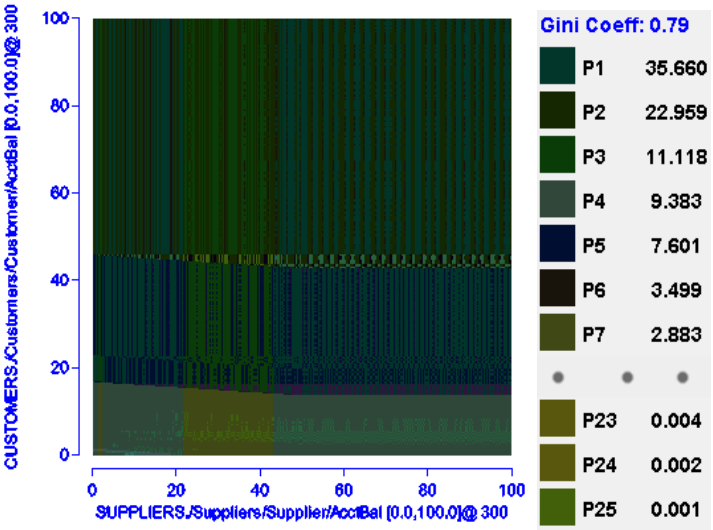
In the plan diagram corresponding to XBench, the maximum Jaccard dissimilarity is be-



(a) **TPoX**



(b) **XBench**



(c) **TPCH_X**

Figure 5.24: Semantically Colored XML Plan Diagrams

Benchmark	No of plans	% of point pairs treated well	% of plan pairs treated well	Lowest Ratio	Highest ratio	Objective with weights	Objective w/o weights
TPoX	14	99.97%	95.6%	1.32	1.34	$1.16E-5$	$1.7E-3$
XBench	42	98.95%	85.5%	0.09	314.17	$1.3E-3$	$6.9E-3$
TPCH_X	25	98.45%	92.0%	0.55	34.65	$4.2E-4$	$1.3E-3$

Table 5.3: Statistics for XML-QTs with weights

tween plans P2 and P8, corresponding to the dark-green sine wave like pattern in middle of the plan diagram and the bottom purple horizontal patch, respectively. The maximum color distance is between plans P9 and P42 (1.09), which also happen to be another pair having the maximum Jaccard dissimilarity of 1.06 in the diagram.

Finally, the statistics for the coloring obtained is as given in Table 5.3. We have achieved a very low objective in all three cases. Also, in the third column - '% of point pairs treated well', we can see that more than 90% of the point pairs are treated well in all three cases. Even the '% of plan pairs treated well' given in the fourth column, shows that a majority of the plan pairs are colored well in all three benchmarks. Thus, the coloring obtained reflects the plan dissimilarities with satisfactory visual accuracy.

Chapter 6

Conclusions

We have investigated in this thesis, three new semantic features of *plan diagrams*, a popular and powerful tool used in the analysis of modern database query optimizers. As the first feature, we introduced and discussed a *structure-based* reduction scheme, as an alternative to the cost-based reduction strategies that had been previously proposed in the literature. In this new reduction scheme, we coalesced all plans with the same join order into one entity, leaving the diagram with as many colors as the number of unique join orders. Structure-based reduction has useful applications in join order caching, when the join order cardinalities of plan diagrams are low or moderate. With join order caching, we always provide the best plan as would have been chosen by the optimizer, in far less time than required for fresh optimization. Additionally, we provide *stable* plan choices, i.e., we always choose the same best plan for a given query point, which is an important requirement in industrial settings. Also, when values of certain parameters cannot be predicted at compile time, but can be known at run-time, join order caching can be used as part of a run-time plan refinement strategy to provide better plan choices. Experimental results obtained by employing structure-based reduction on a suite of plan diagrams, in industrial-strength environments, showed that many plan diagrams feature low and moderate join order cardinalities, thus making join order caching an immediate potent application.

To facilitate join order caching even for plan diagrams with high join order cardinalities, we presented the SRE (Small Relation Elimination) heuristic, which brings down the join order cardinalities by removing smaller sized relations from the join orders. While the plan optimality

is no longer guaranteed with the heuristic, we experimentally evaluated the heuristic on the same suite of plan diagrams and observed that the per-query cost-increase was less than 30% for over 90% of the query locations in the plan diagram and even for the remaining 10%, the cost-increase was within acceptable thresholds. Also, we studied the occurrence of common join orders in the plan diagrams produced by different commercial engines. We found the intersection to be surprisingly sparse, and in fact *disjoint* in most cases.

Structure-based reduction provides a bird's eye view of the differences between plans in the plan diagram. In our second piece of work, we investigated a deeper comparison between plans, based on their complete plan tree structures. Specifically, we developed techniques to color plan diagrams in a *semantically* richer way, so that the differences in colors between any pair of plans, reflected the differences found in their structures. With this new approach to coloring, the plan diagram itself provides a first-cut reflection of the plan-tree differences without having to go through the details of every plan. Towards achieving this goal, we first described methods to quantitatively assign differences between plans and then developed techniques to transform these differences to color space. Specifically, we adapted Kruskal's Iterative Steepest Descent (ISD) method, a multidimensional scaling (MDS) technique, to solve our problem. The adaption involved placing algebraic constraints during the scaling process to accommodate geometric restrictions inherent in the output color space. We provided a detailed experimental evaluation by coloring a suite of plan diagrams using the adapted ISD technique. We found the quality of coloring obtained to be visually satisfactory with more than 85% of the point pairs being treated well (within 30% error threshold) in most plan diagrams. Also, we observed that plans, which are structurally similar to each other generally occupy spaces away from the axes.

As our last semantic feature, we re-engineered the plan diagram notion to the flexibly structured world of XML. We elaborated on the various complexities involved in achieving the goal, which included determination of granularity at which statistics are to be collected (node level or document level) and identifying the validity constraints for an XQuery template. We presented representative plan diagrams for a variety of XML data benchmarks – TPoX, XBench and TPCH_X (XML version of TPC-H benchmark) – on an industrial-strength commercial database engine. The results indicate that XML plan diagrams often feature complicated plan

geometries that remain in the plan diagram even after both cost and structure-based reductions. In fact, for both TPoX and TPCH_X, a very high cost-increase threshold was required to remove the complex plan geometries (to obtain a "cleaner" plan diagram) using cost-based reduction. With respect to applying structure-based reduction on XML plan diagrams, we found the resulting join order cardinalities to be very sparse (in single digits), even when the query templates contained more than 5 relations. This fact along with the observation that more than half of the plans were structurally very similar to each other were reflected in the semantically colored versions of the XML plan diagrams.

In summary, we explored in this thesis a variety of semantic features in plan diagrams, which can aid in the informed analysis and understanding of query optimizer behavior.

6.1 Future Work

1. The empirical performance of the current plan tree differencing metric suits the semantic coloring application for which it is used. It would be interesting and useful to come up with a theoretically sound formulation of a plan tree differencing metric which takes into account "database semantics".

2. As part of the adaption of Kruskal's Iterative Steepest Descent, we enforce the required geometric constraints before moving the vertices in each iteration. However, convergence guarantees are provided when such enforcements are done after moving the vertices in each iteration. This alternative approach could be employed to color plan diagrams and compared with the previous approach.

3. We investigated in this thesis, a new coloring scheme for plan diagrams, where the differences in colors between any pair of plans, reflected the differences found in their structures. A useful alternative would be to color in a visual friendly manner, wherein the plan colors are assigned to maximize their visual separation from their neighbours. That is, the goal is to solve a graph coloring problem wherein the objective is to match proximity in selectivity space to distance in (normalized) RGB color space.

4. The semantic coloring technique explored in this thesis works on ab initio colorings, where the coloring for all plans is done from scratch. However, in practice, given a pair of plan diagrams, it is often found useful to color the plans such that matching plans have identical colors in both diagrams. We intend to devise an efficient algorithm for organically integrating the color consistency requirement while retaining good visualization of the semantic differences.

5. Currently, VSX predicates are defined only on fully enumerated paths. In our future work, we propose to provide support for predicates that include *wild cards* in their element paths.

6. Our analysis of XML plan diagrams was on the DB_XML engine, an early supporter of storing and querying XML data. In recent times, other engines have also extended their support to XML. It would be insightful to obtain and analyze XML plan diagrams on these engines as well.

Bibliography

[1] S. Abiteboul, D. Quass, J. Mchugh, J. Widom and J. Wiener, "The Lorel Query Language for Semistructured Data", *Intl. Journal on Digital Libraries 1(1)*, September 1997.

[2] A. Aboulnaga, A. Alameldeen and J. Naughton, "Estimating the Selectivity of XML Path Expressions for Internet Scale Applications", *Proc. of 27th Intl. Conf. on Very Large Data Bases (VLDB)*, September 2001.

[3] G. Antonshenkov, "Dynamic Query Optimization in Rdb/VMS", *Proc. of 9th IEEE Intl. Conf. on Data Engineering (ICDE)*, April 1993.

[4] A. Balmin, T. Eliaz, J. Hornibrook, L. Lim, G. Lohman, D. Simmen, M. Wang and C. Zhang, "Cost-based optimization in DB2 XML", *IBM Systems Journal 45(2)*, January 2006.

[5] R. Bamford, V. Borkar, M. Brantner, P. Fischer, D. Florescu, D. Graf, D. Kossmann, T. Kraska, D. Muresan, S. Nasoi1 and M. Zacharioudakis, "XQuery Reloaded", *Proc. of the VLDB Endowment (PVLDB) 2(2)*, August 2009.

[6] 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)*, June 2002.

[7] I. Borg and P. Groenen, *Modern Multidimensional Scaling: theory and applications" (2nd ed.)*, Springer, 2005.

[8] D. Chamberlin, J. Robie and D. Florescu, "An XML Query Language for Heterogeneous Data Source", *WebDB (Informal Proceedings)*, May 2000.

[9] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems", *Proc. of seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems (PODS)*, June 1998.

[10] 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.

[11] Pooja N. Darera, "Reduction of Query Optimizer Plan Diagrams", *Master's Thesis, Dept. of SERC, Indian Institute of Science*, `http://dsl.serc.iisc.ernet.in/publications/thesis/pooja.pdf`, August 2007.

[12] G. Das and J. Haritsa, "Robust Heuristics for Scalable Optimization of Complex SQL Queries", *Proc. of 23rd IEEE Intl. Conf. on Data Engineering (ICDE)*, April 2007.

[13] A. Deutsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu, "XML-QL: A query language for XML", *Proc. of 8th Intl. World Wide Web Conf.*, May 1999.

[14] A. Dey, S. Bhaumik, Harish D. and J. Haritsa, "Efficiently Approximating Query Optimizer Plan Diagrams", *Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2008.

[15] P. Gassner, G. Lohman and K. Schiefer, "Query optimization in the IBM DB2 family", *IEEE Data Engineering Bulletin 16(4)*, December 1993.

[16] K. Geng, G. Dobbie and Y. Meng, "Survey of XML Semantic Query Optimization", *Proc. of 2009 Fourth Intl. Conf. on Internet Computing for Science and Engineering (ICICSE)*, December 2009.

[17] A. Ghosh, J. Parikh, V. Sengar and J. Haritsa, "Plan Selection based on Query Clustering", *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2002.

[18] G. Gou and R. Chirkova, "Efficiently Querying Large XML Data Repositories: A Survey", *IEEE Trans. on Knowledge and Data Engineering 19(10)*, October 2007.

[19] G. Gou and R. Chirkova, "XML Query Processing: A Survey", `ftp://ftp.ncsu.edu/pub/tech/2005/TR-2005-22.pdf`, May 2005.

[20] G. Graefe, "Query Evaluation Techniques for Large Databases", *ACM Computing Surveys 25(2)*, June 1993.

[21] J. Haritsa, "Query Optimizer Plan Diagrams: Production, Reduction and Applications", *Proc. of 27rd IEEE Intl. Conf. on Data Engineering (ICDE)*, April 2011.

[22] S. Haw, and G. Rao, "Query Optimization Techniques for XML Databases", *International Journal of Information Technology 2(1)*, 2005.

[23] 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.

[24] Y. Ioannidis, "Query Optimization", *ACM Computing Surveys 28(1)*, March 1996.

[25] Y. Ioannidis, R. Ng, K. Shim and T. Sellis, "Parametric Query Optimization", *Proc. of 18th Intl. Conf. on Very Large Data Bases (VLDB)*, August 1992.

[26] H. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu, "TIMBER: A Native XML Database", *The VLDB Journal 11(4)*, December 2002.

[27] N. Kabra and D. DeWitt, "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1998.

[28] D. Kossmann and K. Stocker, "Iterative dynamic programming: a new class of query optimization algorithms", *ACM Trans. on Database Systems 25(1)*, March 2000.

[29] J. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis", *Psychometrika 29*, 1964.

[30] J. Kruskal, "Nonmetric multidimensional scaling: A numerical method", *Psychometrika 29*, 1964.

[31] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom, "Lore: A Database Management System for Semistructured Data", *ACM SIGMOD Record 26(3)*, September 1997.

[32] J. McHugh and J. Widom, "Query Optimization for XML", *Proc. of 25th Intl. Conf. on Very Large Data Bases (VLDB)*, September 1999.

[33] P. Michiels, "XQuery Optimization", *Proc. of VLDB 2003 PhD Workshop*, September 2003.

[34] G. Moerkotte and T. Neumann, "Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products", *Proc. of 32nd Intl. Conf. on Very Large Data Bases (VLDB)*, September 2006.

[35] M. Nicola, I. Kogan and B. Schiefer, "An XML Transaction Processing Benchmark (TPoX)", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2007.

[36] M. Nicola and B. Linden, "Native XML support in DB2 universal database", *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB)*, August 2005.

[37] K. Ono and G. Lohman, "Measuring the complexity of join enumeration in query optimization", *Proc. of 16th Intl. Conf. on Very Large Data Bases (VLDB)*, August 1990.

[38] 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.

[39] F. Reiss and T. Kanungo, "A Characterization of the Sensitivity of Query Optimization to Storage Access Cost Parameters", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2003.

[40] P. Roy, S. Seshadri, S. Sudarshan and S. Bhobe, "Efficient and Extensible Algorithms for Multi Query Optimization", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 2000.

[41] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu and R. Busse, "XMark: A Benchmark for XML Data Management", *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2002.

[42] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie and T. Price, "Access Path Selection in a Relational Database System", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 1979.

[43] 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.

[44] A. Silberschatz, H. Korth and S. Sudarshan, *Database System Concepts*, McGraw-Hill , 1997.

[45] M. Simalango, "XML Query Processing and Query Languages: A Survey", *Computing Research Repository (CORR) abs/1010.1*, December 2010.

[46] M. Steyvers, "Multidimensional Scaling", *Encyclopedia of Cognitive Science*, 2002

[47] P. Tan, M. Steinbach and V. Kumar, "Introduction to Data Mining", *Addison-Wesley*, 2005.

[48] S. Valluri, K. Karlapalem and A. Hulgeri, "Towards Empirically Driven Query Optimization", *Technical Report, IIIT Hyderabad*, `http://web2py.iiit.ac.in/publications/default/view_publication/techreport/55`, September 2009.

[49] F. Waas and C. Galindo-Legaria, "Counting, enumerating, and sampling of execution plans in a cost-based query optimizer", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 2000.

[50] P. Walmsley, *XQUERY*, O'Reilly Publications, 2007.

[51] 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.

[52] http://dsl.serc.iisc.ernet.in/projects/PICASSO

[53] http://dsl.serc.iisc.ernet.in/projects/PICASSO_XML/index.html

[54] http://en.wikipedia.org/wiki/Bellman_equation

[55] http://en.wikipedia.org/wiki/Level_of_measurement

[56] http://en.wikipedia.org/wiki/Multidimensional_scaling

[57] http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

[58] http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.db2.luw.admin.cmd.doc/doc/r0024085.html

[59] http://tpox.svn.sf.net/viewvc/*checkout*/tpox/TPoX/documentation/TPoX_DataGeneration_v2.0.pdf

[60] http://www.ibiblio.org/xql/xql-proposal.html

[61] http://www.redbooks.ibm.com/redbooks/pdfs/sg247315.pdf

[62] http://www.tpc.org/tpch

[63] http://www.w3.org/TandS/QL/QL98/pp/maier.html

[64] http://www.w3.org/TR/xpath/

[65] http://www.w3.org/TR/xquery/

[66] http://www.w3.org/XML/Query/

Appendix A

SQL templates

```
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_retailprice :varies
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost <= (
        select
            min(ps_supplycost)
        from
            partsupp, supplier, nation, region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'EUROPE'
            and ps_supplycost :varies
    )
order by
    s_acctbal desc,
    n_name, s_name, p_partkey
```

Figure A.1: QT2

```
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= '1994-01-01'
    and o_orderdate < '1995-01-01'
    and c_acctbal :varies
    and s_acctbal :varies
group by
    n_name
order by
    revenue desc
```

Figure A.2: QT5

```
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 r_name = 'AMERICA'
            and s_nationkey = n2.n_nationkey
            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
```

Figure A.3: QT8

```
select
    n_name,
    o_year,
    sum(amount)
from
    (
        select
            n_name,
            YEAR(o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) -
            ps_supplycost * l_quantity as amount
    from
        part,
        supplier,
        lineitem,
        partsupp,
        orders,
        nation
    where
        s_suppkey = l_suppkey
        and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey
        and p_partkey = l_partkey
        and o_orderkey = l_orderkey
        and s_nationkey = n_nationkey
        and p_name like '%green%'
        and s_acctbal :varies
        and ps_supplycost :varies
) as profit
group by
    n_name,
    o_year
order by
    n_name,
    o_year desc
```

Figure A.4: QT9

Appendix B

XQuery templates

```
#NAMESPACE declare default element namespace
"http://www.fixprotocol.org/FIXML-4-4";
#NAMESPACE declare namespace
 c="http://tpox-benchmark.com/custacc";
#PSX /c:Customer/c:Accounts/c:Account/
 c:Balance/c:WorkingBalance CUSTACC CUSTACC CADOC
#PSX /FIXML/Order/OrdQty/@Cash ORDER ORDER ODOC
# -- #

XQUERY
 declare default element namespace
 "http://www.fixprotocol.org/FIXML-4-4";
 declare namespace c="http://tpox-benchmark.com/custacc";
 for  $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")
    /c:Customer/c:Accounts/c:Account[
   c:Balance/c:WorkingBalance :varies]
 for $ord in db2-fn:xmlcolumn("ORDER.ODOC")/FIXML/Order
   [@Acct=$cust/@id/fn:string(.) and OrdQty/@Cash :varies]
 order by $cust/c:AccountTitle/text()

 return
   $<$Customer$>$
     {$cust/c:AccountTitle}
     {$cust/c:Currency}
   $<$/Customer$>$
```

Figure B.1: XQuery Template for TPoX (QTX_SEC)

```
#PSX /order/total ORDER ORDER ORDER
#PSX /customers/customer/address_id CUSTOMER
  CUSTOMER CUSTOMER
#--#
XQUERY
 for $order in db2-fn:xmlcolumn("ORDER.ORDER")/order,
  $cust in db2-fn:xmlcolumn
   ("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>
```

Figure B.2: XQuery Template for XBench (QTX19)

```
#VSX /Suppliers/Supplier/AcctBal
 SUPPLIERS SUPPLIERS SUPPLIERS
#VSX /Customers/Customer/AcctBal
 CUSTOMERS CUSTOMERS CUSTOMERS
# -- #
XQUERY
  for $reg in db2-fn:xmlcolumn("REGIONS.REGIONS")
    /Regions/Region[Name="AFRICA"]
    for $nat in db2-fn:xmlcolumn("NATIONS.NATIONS")
      /Nations/Nation[xs:double(RegionKey)=$reg/
        @key/xs:double(.)]
    let $sup := db2-fn:xmlcolumn("SUPPLIERS.SUPPLIERS")
      /Suppliers /Supplier[(xs:double(NationKey)=$nat/
       @key/xs:double(.)) and  (AcctBal :varies)],
    $cust := db2-fn:xmlcolumn("CUSTOMERS.CUSTOMERS")
      /Customers/Customer[(xs:double(NationKey)=$sup/
       NationKey/xs:double(.)) and  (AcctBal :varies)],
    $line := db2-fn:xmlcolumn("ORDERS.ORDERS")/Orders/
      Order[xs:double(CustKey) = $cust/@key/xs:double(.)
       and OrderDate[. > "1994-01-01" and
       . < "1997-01-01"]]/LineItem[xs:double(SuppKey)
        = $sup/@Key/xs:double(.)]
    let $summand :=(for $x in $line
       return $x/ExtendedPrice*(1 -$x/Discount))
    order by sum($summand)

    return
       <result> {$nat/@key} {$nat/Name}
       <revenue> {sum($summand)} </revenue>
       </result>
```

Figure B.3: XQuery Template for TPCH_X

Appendix C

Sample TPCH_X documents

```
<Nations>
  <Nation key="0">
    <Name> ALGERIA </Name>
    <RegionKey> 0 </RegionKey>
    <Comment> ruthlessly bold dinos across </Comment>
  </Nation>
</Nations>
```

(a) **NATIONS**

```
<Regions>
  <Region key="0">
    <Name> AFRICA </Name>
    <Comment> sly, fluffy somas within the daringly thin
      warhorses sublat </Comment>
  </Region>
</Regions>
```

(b) **REGIONS**

```
<Customers>
  <Customer key="000000001" >
    <Name> Customer#000000001 </Name>
    <Address> eCDWmkYrnh5pTZhNEoZbPq </Address>
    <NationKey> 23 </NationKey>
    <Phone> 33-716-804-6669 </Phone>
    <AcctBal> 8409.98 </AcctBal>
    <MktSegment> MACHINERY </MktSegment>
    <Comment> fluffily silent hockey </Comment>
  </Customer>
</Customers>
```

(c) **CUSTOMERS**

```
<Suppliers>
  <Supplier Key="0000000001">
    <Name> Supplier#0000000001 </Name>
    <Address> hONA5x_e(Iq# AqJ{Bb+Pi$hM </Address>
    <NationKey> 21 </NationKey>
    <Phone> 31-551-967-3671 </Phone>
    <AcctBal> 4167.78 </AcctBal>
    <Comment> waters x-ray platelets–dare </Comment>
  </Supplier>
</Suppliers>
```

(d) **SUPPLIERS**

```
<Parts>
  <Part>
    <PartKey> 1 </PartKey>
    <Name> cream sandy forest misty papaya </Name>
    <Mfgr> Manufacturer#3 </Mfgr>
    <Brand> Brand#33 </Brand>
    <Type> STANDARD PLATED BRASS </Type>
    <Size> 4 </Size>
    <Container> MED PACK </Container>
    <RetailPrice> 901.00 </RetailPrice>
    <P_Comment> ironic epita </P_Comment>
    <PartSupp>
      <Suppkey> 2 </Suppkey>
      <Avail> 4625 </Avail>
      <SupplyCost> 90.48 </SupplyCost>
      <PS_Comment> sheaves toward the blithe, thin frays
        promise doggedly final, final dugouts;
        ironic </PS_Comment>
    </PartSupp>
    <PartSupp>
      <Suppkey> 252 </Suppkey>
      <Avail> 6437 </Avail>
      <SupplyCost> 965.89 </SupplyCost>
      <PS_Comment> regular escapades behind the
        thin, silent patterns may boost ruthlessly for
        the clos </PS_Comment>
    </PartSupp>
    <PartSupp>
      <Suppkey> 502 </Suppkey>
      <Avail> 9158 </Avail>
      <SupplyCost> 454.60 </SupplyCost>
      <PS_Comment> Tiresias would kindle regularly even
        dugouts!fluffy, busy gifts on the somas might sleep
        around the ironically furious dependencies.busy, sly
        patterns along t </PS_Comment>
    </PartSupp>
  </Part>
</Parts>
```

(e) **PARTS**

```
<Orders>
  <Order>
    <OrderKey> 1 </OrderKey>
    <CustKey> 1</CustKey>
    <OrderStatus> F </OrderStatus>
    <TotalPrice> 68113.8155 </TotalPrice>
    <OrderDate> 1996-02-16 </OrderDate>
    <OrderPriority> 3-MEDIUM </OrderPriority>
    <Clerk> Clerk#000000994 </Clerk>
    <ShipPriority> 0 </ShipPriority>
    <O_Comment> ruthless dependencies </O_Comment>
    <LineItem>
      <PartKey> 1 </PartKey>
      <SuppKey> 502 </SuppKey>
      <LineNumber> 1 </LineNumber>
      <Quantity> 38 </Quantity>
      <ExtendedPrice> 34238.0000 </ExtendedPrice>
      <Discount> 0.0992 </Discount>
      <Tax> 0.0393 </Tax>
      <ReturnFlag> R </ReturnFlag>
      <LineStatus> F </LineStatus>
      <ShipDate> 1996-03-09 </ShipDate>
      <CommitDate> 1996-05-06 </CommitDate>
      <ReceiptDate> 1996-03-11 </ReceiptDate>
      <ShipInstruct> NONE </ShipInstruct>
      <ShipMode> SHIP </ShipMode>
      <L_Comment> ironically brave theodoli </L_Comment>
    </LineItem>
    <LineItem>
      <PartKey> 2 </PartKey>
      <SuppKey> 753 </SuppKey>
      <LineNumber> 2 </LineNumber>
      <Quantity> 2 </Quantity>
      <ExtendedPrice> 1804.0000 </ExtendedPrice>
      <Discount> 0.0923 </Discount>
      <Tax> 0.0144 </Tax>
      <ReturnFlag> A </ReturnFlag>
      <LineStatus> F </LineStatus>
      <ShipDate> 1996-04-22 </ShipDate>
      <CommitDate> 1996-04-07 </CommitDate>
      <ReceiptDate> 1996-05-03 </ReceiptDate>
      <ShipInstruct> DELIVER IN PERSON </ShipInstruct>
      <ShipMode> SHIP </ShipMode>
      <L_Comment> final, permanent warho </L_Comment>
    </LineItem>
  </Order>
</Orders>
```

(f) **ORDERS**

Figure C.1: Sample XML Documents for TPCH_X Schema