Next Generation Relational Query Optimizers

A Project Report

Submitted in Partial Fulfilment of the

Requirements for the Degree of

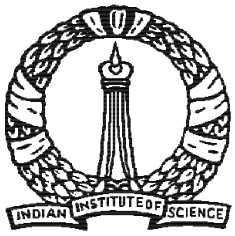
Master of Engineering

in

Computer Science and Engineering

by

Naveen Reddy



Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012

June 2005

to my parents

without whose constant support and urging

I would never have made it this far.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to begin by thanking my guide Prof. Jayant Haritsa for his enduring support and encouragement as well as the freedom to pursue my area of interest. I would also like to thank my fellow ME-DSL'ite Pavan for his wonderful company and help. Last, but not least, I thank all my friends who directly or indirectly helped me in completing this report.

Abstract

Plastic [7] is a recently-proposed tool to help query optimizers significantly amortize optimization overheads through a technique of plan recycling. This tool groups similar queries into clusters and uses the optimizer-generated plan for the cluster representative to execute all future queries assigned to the cluster. We have now significantly extended the scope, useability and efficiency of Plastic by incorporating a host of new features, including *dynamic variable-sized* clustering with L^∞*Norm* as the distance metric, *decision-tree-based* query classifier, integrated *plan-cost* and *plan-cardinality* diagrams and an extended *plan-diff* module that considers *operator-specific* details and can compare plans across database systems. Plastic which originally worked on DB2 and Oracle has now been ported to *MS-SQL Server* and *PostgreSQL*.

Plastic reduces optimization overheads by amortizing the cost of producing an optimal plan. Picasso [18], a tool conceived and developed by us, can help in reducing the cost of producing the optimal plan itself. In this context, we define a *plan diagram* [7] as a pictorial enumeration of the execution plan choices of a database query optimizer over the relational selectivity space. Using Picasso, we present and analyze representative plan diagrams on a suite of popular commercial query optimizers for queries based on the TPC-H benchmark. These diagrams, which often appear similar to cubist paintings, provide a variety of interesting insights, including that current optimizers make extremely fine-grained plan choices, which may often be supplanted by less efficient options without substantively affecting the quality; that the plan optimality regions may have highly intricate patterns and irregular boundaries, indicating strongly non-linear cost models; that non-monotonic cost behavior exists where increasing result cardinalities decrease the estimated cost; and, that the basic assumptions underlying the research literature on parametric query optimization often do not hold in practice. We hope that these inputs can be

used to simplify the design of modern query optimizers and thereby reduce query optimization overheads.

Chapter 1

Introduction

Modern database systems use a *query optimizer* to identify the most efficient strategy to execute the SQL queries that are submitted by users. The efficiency of these strategies, called "plans", is usually measured in terms of query response time. Optimization is a mandatory exercise since the difference between the cost of the best execution plan and a random choice could be in orders of magnitude. The role of query optimizers has become especially critical in recent times due to the high degree of query complexity characterizing current data warehousing and mining applications, as exemplified by the TPC-H decision support benchmark [45].

While commercial query optimizers each have their own "secret sauce" to identify the best plan, the de-facto standard underlying strategy, based on the classical System R optimizer [24], is the following: Given a user query, apply a variety of heuristics to restrict the combinatorially large search space of plan alternatives to a manageable size; estimate, with a cost model and a dynamic-programming-based processing algorithm, the efficiency of each of these candidate plans; finally, choose the plan with the lowest estimated cost.

These well known inherent costs are compounded by the fact that a new query submitted to the database system is typically optimized afresh, providing no opportunity to amortize these overheads over prior optimizations. While current commercial query optimizers do provide facilities for reusing execution plans (e.g. "stored outlines" in Oracle [41]), the query matching is extremely restrictive – only if the incoming query has a close *textual resemblance* with one of the stored queries then the associated plan is reused to execute the new query.

The Plastic Tool

Recently, in [7], a tool called **Plastic** (PLan Selection Through Incremental Clustering) was proposed to be used by query optimizers to significantly increase the scope of plan reuse and amortize the optimization overheads through a technique of "plan recycling". The tool is based on the observation that queries which differ in projection, selection, join predicates, and even in the base tables themselves, may still have identical *plan templates* – that is, they share a common database operator tree. By identifying such similarities in the plan space, Plastic materially improves the utility of plan cacheing.

Specifically, Plastic attempts to capture these similarities by characterizing queries in terms of a feature vector that includes structural attributes such as the number of tables and joins in the query, as well as statistical quantities such as the sizes of the tables participating in the query. Using a distance function defined on these feature vectors, queries are grouped into clusters. Clusters are built incrementally using the *leader* algorithm proposed by Hartingan [10]. Each cluster has a representative for whom the template of the optimizer-generated execution plan is persistently stored, and this plan template is used to execute all future queries assigned to the cluster. In short, Plastic recycles plan templates based on the expectation that its clustering mechanism is likely to assign an execution plan that is identical to what the optimizer would have produced on the same query. A sample output depicting clustering in Plastic for *query template*[1] Q2, is shown in Figure 1.1. Here, the axes represent the selectivities of a pair of the participating relations, namely PART and PARTSUPP and the dots represent the cluster representatives (leaders). Experiments with a variety of TPC-H based queries on a commercial optimizer showed that Plastic predicts the correct plan choice in most cases, thereby providing significantly improved query optimization times. Further, even when errors were made, the additional execution cost incurred due to the sub-optimal plan choices was marginal.

Apart from the obvious advantage of speeding up the optimization, Plastic also improves query execution efficiency since it makes it possible for optimizers to always run at their highest optimization level as the cost of such optimization is amortized over all future queries that reuse

[1] A query template represents a query in which some or all of the constants in the where-clause predicates have been replaced by bind variables.

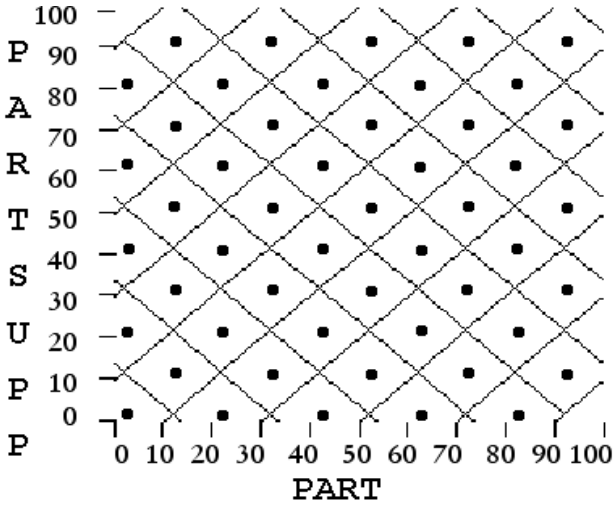


Figure 1.1: Fixed-Sized clustering

these plans. Yet another important advantage is that the benefits of "plan hints", a common technique for influencing optimizer plan choices for specific queries, automatically percolate to the entire set of queries that are associated with this plan. Lastly, since the assignment of queries to clusters is completely based on database statistics, the plan choice for a given query is *adaptive* to the current state of the database.

Earlier demos [26, 23] had presented two implementations of Plastic, one that demonstrated the basic prototype, and another which improved upon the first. We have now significantly extended the scope, useability, and efficiency of Plastic, by incorporating a host of new features, including:

- Implementation of Plastic in MS-SQL Server.
- Implementation of Plastic in PostgreSQL.
- L^∞*Norm* as the distance metric to better match boundaries in the plan space.
- *Dynamic variable-sized* clustering to match volatility in plan space.
- Integration of a C4.5 decision tree classifier for fast cluster identification.
- An extended *plan-diff* module that compares plans considering the *operator-level* attributes and can compare plans across database systems.

Plan and Cost Diagrams

For a query on a given database and system configuration, the optimal plan choice is primarily a function of the *selectivities* of the base relations participating in the query – that is, the estimated number of rows of each relation relevant to producing the final result. In this report, we use the term "plan diagram" to denote a color-coded pictorial enumeration of the execution plan choices of a database query optimizer over the relational selectivity space. An example two dimensional plan diagram is shown in Figure 1.2(a), for a query based on Query 7 of the TPC-H benchmark, with selectivity variations on the ORDERS and CUSTOMER relations[2].

[*Note to Readers: We recommend viewing all diagrams presented in this paper directly from the color PDF file, available at [32], or from a color print copy, since the greyscale version may not clearly register the various features.*]

In this picture, produced with a popular commercial query optimizer, a set of six optimal[3] plans, P1 through P6, cover the selectivity space. The value associated with each plan in the legend indicates the percentage space coverage of that plan – P1, for example, covers about 38% of the area, whereas P6 is chosen in only 1% of the region.

Complementary to the plan diagram is a "cost diagram", shown in Figure 1.2(b), which is a three-dimensional visualization of the estimated plan execution costs over the same relational selectivity space (in this picture, the costs are normalized to the maximum cost over the space, and the colors reflect the relative magnitude with blue indicating low cost, white – medium cost, and red – high cost.)

The Picasso Tool

Plastic reduces optimization overheads by amortizing the cost of producing an optimal plan – that is, in a given cluster, once the optimizer generates a plan for the *leader*, Plastic optimizes the cost of generating plans for the *followers*. On the other hand, Picasso [18], a tool conceived and developed by us, can help in reducing the *cost* of producing the *optimal plan* for the leader.

[2] Specifically, the variation is on the `o_totalprice` and `c_acctbal` attributes of these relations.
[3] The optimality is with respect to the optimizer's restricted search space, and not in a global sense.

In certain cases, as detailed later, Picasso can also help in improving the *quality* of the *optimal plan* itself.

Given a query and a relational engine, Picasso automatically generates the associated plan and cost diagrams. In this report, we describe the plan/cost/cardinality diagrams output by Picasso on a suite of three popular commercial query optimizers for queries based on the TPC-H benchmark. [Due to legal restrictions, these optimizers are randomly identified as OptA, OptB and OptC, in the sequel.]

Our evaluation shows that a few queries in the benchmark do produce "well-behaved" or "smooth" plan diagrams like that shown in Figure 1.2(a). A substantial remainder, however, result in complex and intricate plan diagrams that appear similar to *cubist paintings* [35][4], providing rich material for investigation. A particularly compelling example is shown in Figure 1.3(a) for Query 8 of the TPC-H benchmark with OptA[5], where no less than 68 plans cover the space in a highly convoluted manner! Further, even this cardinality is a *conservative* estimate since it was obtained with a query grid of 100 x 100 – a finer grid size of 300 x 300 resulted in the plan cardinality going up to 80 plans!

Before we go on, we hasten to clarify that our goal in this paper is to provide a broad overview of the intriguing behavior of modern optimizers, but *not* to make judgements on specific optimizers, nor to draw conclusions about the relative qualities of their execution plans. Further, not being privy to optimizer internals, some of the conclusions drawn here are perforce speculative in nature and should therefore be treated as such. Our intention is primarily to alert database system designers and developers to the phenomena that we have encountered during the course of our study, with the hope that they may prove useful in building the next generation of optimizers.

Analysis of Plan and Cost Diagrams

Analyzing the TPC-H based query plan and cost diagrams provides a variety of interesting insights, including the following:

[4]Hence, the name of our tool – Pablo Picasso is considered to be a founder of the cubist painting genre [35].
[5]Operating at its highest optimization level.

Fine-grained Choices: Modern query optimizers often make extremely *fine-grained* plan choices, exhibiting a marked skew in the space coverage of the individual plans. For example, 80 percent of the space is usually covered by less than 20 percent of the plans, with many of the smaller plans occupying less than *one percent* of the selectivity space. Using the well-known Gini index [34], which ranges over [0,1], to quantify the skew, we find that all the optimizers, *across the board,* exhibit a marked skew in excess of 0.5 for most queries, on occasion going even higher than 0.8.

Further, and more importantly, we show that the small-sized plans may often be supplanted by larger siblings *without substantively affecting the quality*. For example, the plan diagram of Figure 1.3(a) which has 68 plans can be "reduced" to that shown in Figure 1.3(b) featuring as few as *seven* plans, without increasing the estimated cost of any individual query point by more than 10 percent.

Overall, this leads us to the hypothesis that current optimizers may perhaps be oversophisticated in that they are "doing too good a job", not merited by the coarseness of the underlying cost space. Moreover, if it were possible to simplify the optimizer to produce only reduced plan diagrams, it is plausible that the considerable processing overheads typically associated with query optimization could be significantly lowered.

Complex Patterns: The plan diagrams exhibit a variety of intricate tessellated patterns, including *speckles, stripes, blinds, mosaics* and *bands,* among others. For example, witness the rapidly alternating choices between plans P12 (dark green) and P16 (light gray) in the bottom left quadrant of Figure 1.3(a). Further, the boundaries of the plan optimality regions can be highly irregular – a case in point is plan P8 (dark pink) in the top right quadrant of Figure 1.3(a). These complex patterns appear to indicate the presence of strongly non-linear and discretized cost models, again perhaps an over-kill in light of Figure 1.3(b).

Non-Monotonic Cost Behavior: We have found quite a few instances where, although the base relation selectivities and the result cardinalities are monotonically increasing, the

cost diagram does *not* show a corresponding monotonic behavior.[6] Sometimes, the non-monotonic behavior arises due to a change in plan, perhaps understandable given the restricted search space evaluated by the optimizer. But, more surprisingly, we have also encountered situations where a plan shows such behavior even *internal* to its optimality region.

Validity of PQO: A rich body of literature exists on *parametric query optimization* (PQO) [1, 2, 11, 12, 5, 6, 14, 16, 17]. The goal here is to apriori identify the optimal set of plans for the entire relational selectivity space at compile time, and subsequently to use at run time the actual selectivity parameter settings to identify the best plan – the expectation is that this would be much faster than optimizing the query from scratch. Much of this work is based on a set of assumptions, that we do not find to hold true, *even approximately*, in the plan diagrams produced by the commercial optimizers.

For example, one of the assumptions is that a plan is optimal within the *entire region* enclosed by its plan boundaries. But, in Figure 1.3(a), this is violated by the small (brown) rectangle of plan P14, close to coordinates (60,30), in the (light-pink) optimality region of plan P3, and there are several other such instances.

On the positive side, however, we show that some of the important PQO assumptions do hold approximately for *reduced* plan diagrams.

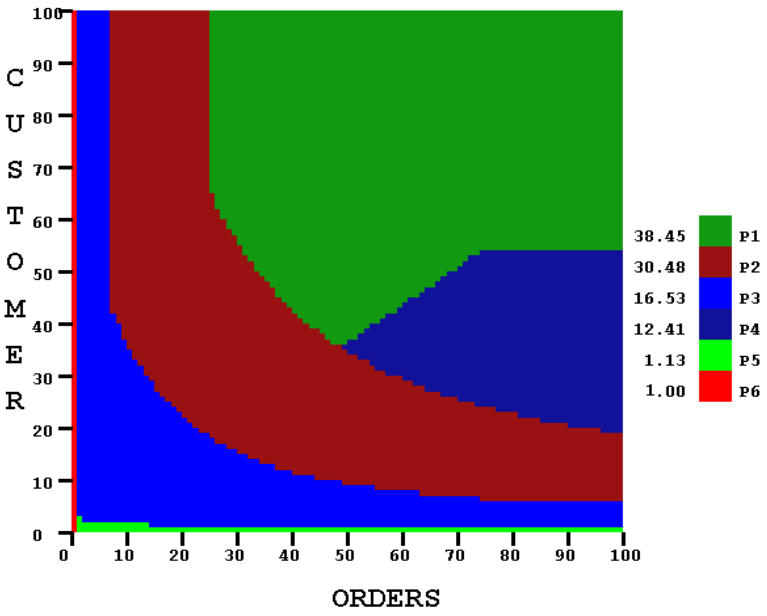
1.1 Organization

In this report we present a walk-through of the upgraded Plastic tool, and explain how it helps to significantly amortize the overheads of query optimization. We also present the Picasso tool, and show how it serves as a research, educational and administrative tool for understanding the intricacies of query plan generation.

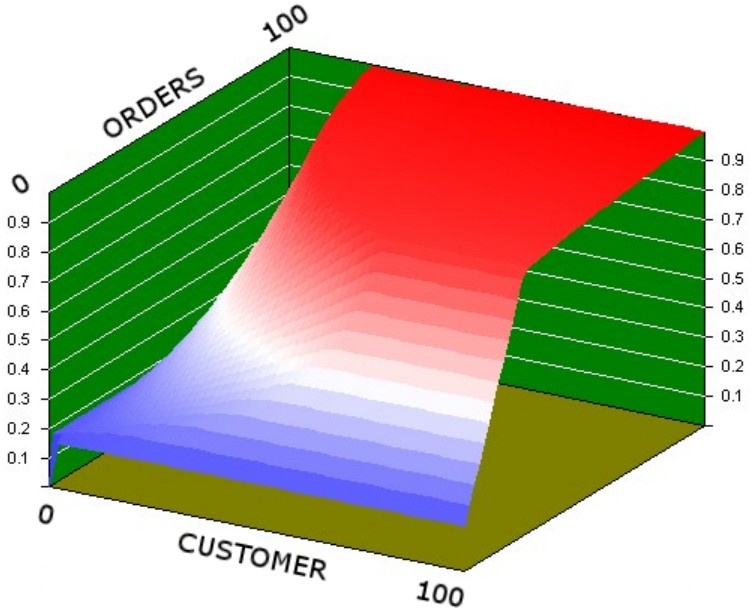
The remainder of this report is organized as follows: In Section 2, we discuss related work.

[6] Our query setup is such that in addition to the result cardinality monotonically increasing as we travel outwards along the selectivity axes, the result tuples are also *supersets* of the previous results.

Integrated architecture of Plastic and Picasso are discussed in Section 3. The new features incorporated into Plastic are discussed in Section 4. The Picasso tool and the testbed environment for our experiments is presented in Section 5. Then, in Section 6, the skew in the plan space distribution, as well as techniques for reducing the plan set cardinalities, are discussed. The relationship to PQO is explored in Section 7. Interesting plan motifs are presented in Section 8. Finally, in Section 9, we summarize the conclusions of our study and outline future research avenues.

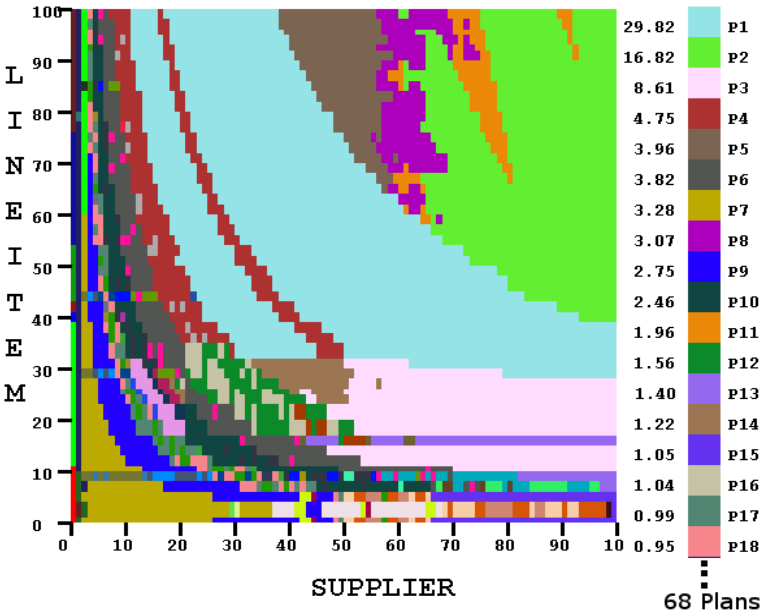


(a) **Plan Diagram**

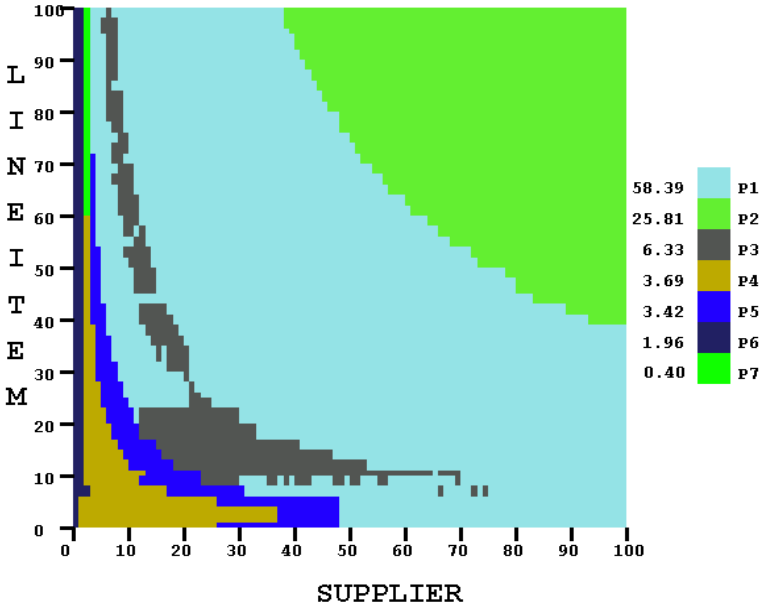


(b) **Cost Diagram**

Figure 1.2: Smooth Plan and Cost Diagram (Query 7)



(a) **Complex Plan Diagram**



(b) **Reduced Plan Diagram**

Figure 1.3: Complex Plan and Reduced Plan Diagram (Query 8, OptA)

Chapter 2

Related Work

2.1 Related Work for Plastic

Techniques such as *multi-query optimization* (MQO) [21, 27, 25, 8, 15] and *parametric query optimization* (PQO) [14, 2, 6, 5, 11, 12, 1, 17, 16] have been previously proposed for enhancing the query optimization process. Both these techniques are inherently computationally hard – for example, the search space in MQO is doubly exponential in the size of the queries. This has led to the design of heuristic-based solutions, such as those presented in [21].

Plastic approach is fundamentally different from MQO in that it do not attempt to *optimize* queries but merely to make effective use of the *results* of prior optimizations. Moreover, while Plastic does group queries into clusters, the plan selection is applicable on a per-query basis and is therefore not restricted to query batches. Finally, Plastic optimization is not limited to a temporal window of queries, but can be utilized across widely dispersed query sets. Moving on to PQO, its coverage of the query space is typically an off-line process. In contrast, Plastic approach can be implemented in either an off-line manner where artificial queries are generated so as to create clusters that cover the query space, or more practically as an online process with regard to both cluster formation and query plan selection. That is, the query space can be covered incrementally on demand when user queries arrive at the database system. Another significant difference with PQO is that Plastic plan selection process involves only the traversal of a simple decision tree, whereas PQO requires a spatial storage and indexing mechanism. This

is because the scheme requires storing not only the set of optimal plans but also the regions in which each of these plans are optimal. Even for simple linear cost functions, the shapes of these regions turn out to be convex polyhedrons [5], mandating spatial storage and access in order to identify which plan is to be utilized for a newly arrived query. This issue assumes importance since supporting spatial databases is well-known to be an expensive proposition [29]. Finally, while PQO is concerned with completely characterizing the plan space for a given query, our approach extends to *sharing* of plans across similar queries.

2.1.1 Comparison with Modern Optimizers

Some modern optimizers also provide plan reuse facilities. We discuss Oracle9i Optimizer [41] here and how adding Plastic, to any such optimizer, can augment its capabilities. The Oracle database system provides a mechanism, called "stored outlines", for preserving queries and execution plans. When the system parameter USE_STORED_OUTLINES is set to true, the optimizer compares the incoming query with the stored queries and if an *identical* match is found, the associated plan is used. The point to note here is that the query matching is done at the *syntactic* level. There needs to be a one-to-one correspondence between SQL text and its stored outline. If a different literal is specified in a predicate, then a different outline applies. To avoid this, Oracle also allows bind variables to be used instead of constants to allow a wider coverage. This approach is still somewhat limited in several ways. Firstly, the query matching is very strict – a slight change in the structure of query, for example, adding or replacing of a projection attribute, will result in the optimizer not utilizing the existing plan. Secondly, it does not take into account the fact that several selection predicates on a particular table can together generate a selectivity for the table which is similar to that of a previously stored query. Thirdly, a more serious problem is that the query plan is the same for the *complete range of values* of a bind variable since Oracle adopts the heuristic of assuming small values for the selectivity of bind variable-based predicates. Specifically, it chooses a selectivity of 0.05 for all range predicates associated with bind variables, a heuristic that can prove very costly for database environments with higher selectivity values. Plastic approach, on the other hand, tries to address all these three issues in a much more flexible and fine-grained manner.

It should thus be noted that Plastic does not just map a parametric space based on changes in bind variables of selection predicates but works at the level of sharing *between* queries, a feature expected to be *desirable* in practice.

2.2 Related Work for Picasso

To the best of our knowledge, there has been no prior work on the analysis of plan diagrams with regard to *real-world industrial-strength* query optimizers. However, similar issues have been studied in the parametric query optimization (PQO) literature in the context of simplified self-crafted optimizers. Specifically, in [1, 16, 17], an optimizer modeled along the lines of the original System R optimizer [24] is used, with the search space restricted to left-deep join trees, and the workload comprised of pure SPJ queries with "star" or "linear" join-graphs. The metrics considered include the cardinality and spatial distribution of the set of optimal plans – while [1] considered only single-relation selectivities, [16, 17] evaluated two-dimensional relational selectivity spaces, similar to those considered in this paper. Their results in the 2-D case indicate that for linear queries, the average number of optimal plans is linear in the number of join relations, while for star queries, this number is almost quadratic. Also, the optimal plans are found to be densely packed close to the origin and the selectivity axes. An analysis of plan reduction possibilities in [1], given a plan optimality tolerance threshold, indicates that a larger fraction of plans can be removed with increasing query complexity. In [11, 12], an optimizer modeled along the lines of the Volcano query optimizer [9] is used, and they find the cardinality of the optimal plan set for queries with two, three and four-dimensional relational selectivities. They also present efficient techniques for approximating the optimal plan set. Finally, a complexity analysis of the optimal plan set cardinality is made in [5] for the specific case of linear (affine) cost functions in two parameters.

While the above efforts do provide important insights, the results presented in this paper indicate that plan diagrams with sophisticated real-world optimizers and queries show much more variability with regard to both plan set cardinalities and spatial distributions, as compared to those anticipated from the PQO literature. For example, as mentioned earlier, we find that

plan densities can be high even in regions far from the plan diagram axes, and that the optimality region geometries can have extremely irregular boundaries.

There has also been work on characterizing the sensitivity of query optimization to storage access cost parameters [19], but this work focuses on the robustness of optimal plan choices to inaccuracies in the optimizer input parameters, and when suboptimal choices are made, the impact of these errors. So, the focus is on plan *quality*, not quantity or spatial distribution. Further, their analysis shows that when all tables and indexes are on a single device (as in our case), the optimizer proved relatively insensitive to inaccurate resource costs in terms of plan choices – however, we find strong sensitivity with regard to *selectivity values*. Further, many of the queries for which they did find some degree of sensitivity also feature in our list of "dense" queries.

Cost-based attempts to reduce the optimizer's search space include a "pilot-pass" approach [20], where a complete plan is initially computed and the cost of this plan is used to prune the subsequent dynamic programming enumeration by removing all subplans whose costs exceed that of the complete plan. But, it has been reported [13] that such pruning has only marginal impact in real-world environments. Finally, a preliminary study of a sampling-based approach to find acceptable quality plans, evaluated on a commercial optimizer, is discussed in [30], but its impact on the *optimal* plan set cardinality is an open issue.

Chapter 3

Architecture of Plastic and Picasso

A block-level diagram that shows the integrated architecture of Plastic and Picasso is shown in Figure 3.1. In this picture, the solid lines show the sequence of operations in the situation where a matching cluster or a matching plan diagram is found for the new query, while the dashed lines represent the converse situation where no match is available and a fresh cluster or a new plan diagram has to be created.

3.1 Architecture of Plastic

The query given to Plastic is first processed by the *Feature Vector Extractor* which also accesses the system catalogs and obtains the information required to produce the feature vector. The *Similarity Check* module establishes whether this feature vector has a sufficiently close match with any of the cluster representatives in the *Query Cluster Database*. If a match is found (solid lines in Figure 3.1), the plan template for the matching cluster representative is accessed from the *Plan Template Database*. A plan template is a plan that has database operators but does not have the specific values of the inputs like table names, index names to these operators. This plan template is converted into a complete plan by the *Plan Generator* module, which fills in the operator inputs based on the specifics of the input query.

On the other hand, if no matching cluster is found (dashed lines in Figure 3.1), the Query Optimizer is invoked in the traditional manner and the plan it generates is used for executing the

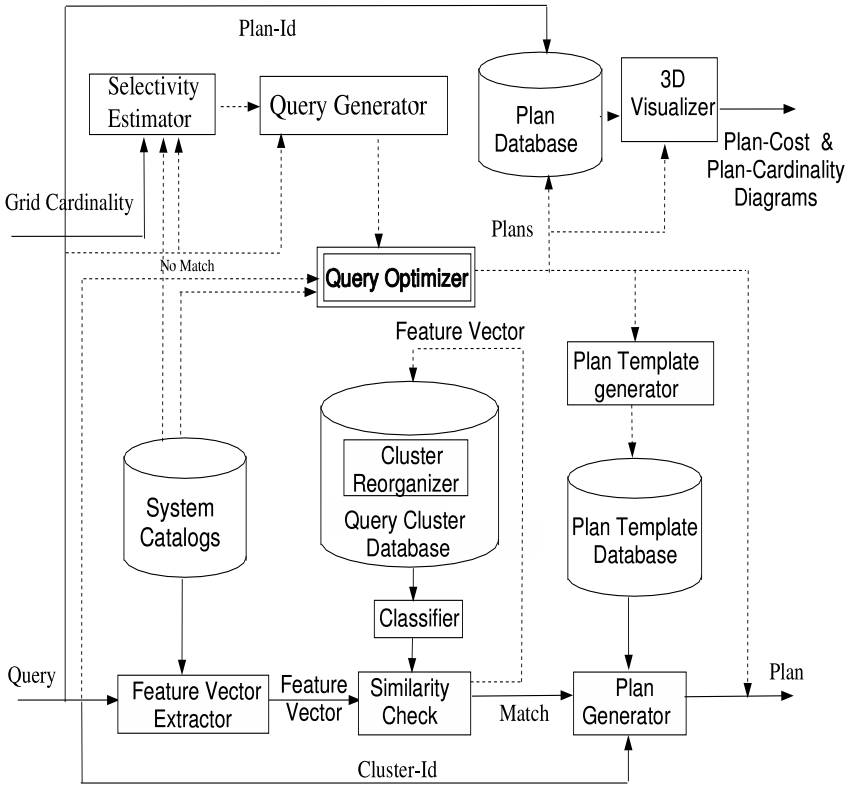


Figure 3.1: Integrated Architecture of Plastic and Picasso

query. This plan is also passed to the *Plan Template Generator* which converts the plan into its template representation and stores it in the *Plan Template Database*. Concurrently, the feature vector of the query is stored in the *Query Cluster Database*, as a new cluster representative. Periodically, the cluster database may be reorganized to suit constraints such as a memory budget or a ceiling on the number of clusters. For example, it may be decided to purge the feature vectors and plan templates of "outlier" queries that rarely result in matches with the current query workload.

3.2 Architecture of Picasso

Picasso takes as input the query, the grid granularity at which queries should be distributed across the plan space, the relations (axes) on which the selectivities should be varied to construct

the plan diagram (can be deduced from the query itself if there are only two relations in the query), and the choice of query optimizer.

The query and the grid granularity given to Picasso is first processed by the *Selectivity Estimator* module which also accesses the system catalogs and obtains the information required to produce the conditions on the chosen two relations for achieving a particular selectivity. These conditions are given to the *Query Generator* module that generates the query for a specified value of selectivity. This query is fed to the *Query Optimizer* that generates a plan to execute the query. Once plans have been generated for each of the points at the specified granularity, the entire set of plans with their corresponding positions in the plan space are persistently stored in the *Plan Database* so that the next time when the same plan diagram is needed, it does not need to be generated. These set of plans and their corresponding positions are also given as input to a *3D Visualizer* module that generates both plan-cost and plan-cardinality diagrams.

Chapter 4

New Features in Plastic

In this section, we describe in detail the new features of Plastic that have been implemented. These new features significantly extend the scope, useability, and efficiency of Plastic as explained below.

4.1 MS-SQL Server and PostgreSQL

Plastic originally worked on DB2 and Oracle. We have ported it to work on *MS-SQL Server* and *PostgreSQL* . Thus Plastic now works on all popular database platforms. With this implementation, we now have a tool that can be used to compare the plans generated for queries across all popular database platforms. This feature of Plastic is very useful, especially in the industry where it could be used to identify areas of the plan space where an inferior plan is being chosen by one optimizer while a relatively better plan is being chosen by another optimizer. Further, it provides a base that can be used to compare the quality of plans produced by different optimizers, having same database and system configuration.

4.2 Clustering in Plastic

For comparing various mechanisms in clustering, we use *query template* Q-21 from the TPC-H benchmark (the nested sub-query and group-by operations have been removed). Here :1 and :2

are “bind variables” that are replaced by constants in an actual query. The query template is given in Figure 4.1.

```
select * from
    supplier,
    lineitem,
    orders,
    nation
where
    s_suppkey = l_suppkey
    and o_orderkey = l_orderkey
    and s_nationkey = n_nationkey
    and l_quantity = : 1
    and n_regionkey = : 2
```

Figure 4.1: Query-21 (template) of TPC-H Benchmark

4.2.1 Static Variable-Sized Clustering

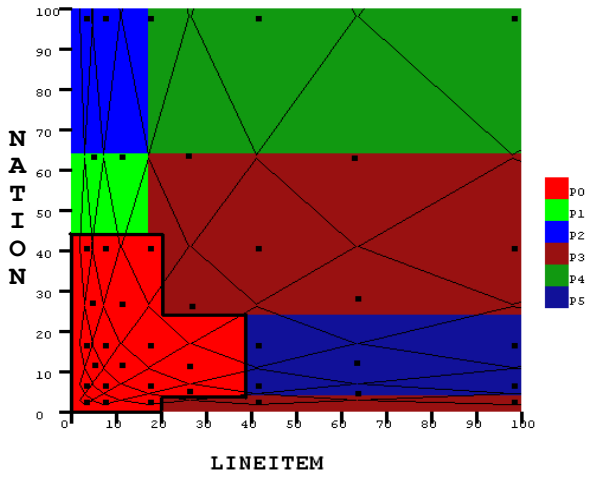


Figure 4.2: Static Variable-Sized clustering

Plastic originally implemented *fixed-size* clusters. The distance metric employed is shown below.

$$Distance(Q_1, Q_2) = \sum_{i=1}^{K} \frac{\mid ETS_1^i - ETS_2^i \mid}{max(TS_1^i, TS_2^i)} \tag{4.1}$$

where Q_1 and Q_2 are queries involving K tables and TS_1^i, ETS_1^i is original and (estimated) effective size of i^{th} table in Q_1. Similar explanation holds for TS_2^i, ETS_2^i.

Fixed-size clustering resulted in the twin problems of insufficient clusters in the *high-volatility* (rapid changes in plan choices) regions of the plan space and redundant clusters in the *low-volatility* (gradual changes in plan choices) regions.

To overcome this, *static variable-sized* clustering was implemented in Plastic [22], which provided several small-sized clusters in the high-volatility region and a few large-sized clusters in the low-volatility region. This scheme was based on the assumption that the high-volatility region is typically present in the highly-selective region of the plan space and the low-volatility region is typically present in the low-selectivity region. The modified distance metric is shown below.

$$Distance(Q_1, Q_2) = \sum_{i=1}^{K} \frac{\mid ETS_1^i - ETS_2^i \mid}{max(ETS_1^i, ETS_2^i)} \tag{4.2}$$

A sample output of static variable-sized clustering for query template Q-21 for the DB2 optimizer, is shown in Figure 4.2. Here, the axes represents the selectivities of a pair of the participating relations, namely NATION and LINEITEM, the dots represent the cluster representatives, and the background shows the associated plan diagram.

Though this technique improves the accuracy of clustering, it still has some disadvantages.

- Firstly note that the L^1Norm (also known as *Manhattan* distance metric) is not a good metric for clustering because, the *plan space* generally has *horizontal* boundaries rather than *diagonal* boundaries, whereas with L^1Norm we get *diagonal* boundaries.

- The observation that the high-volatility region is typically present in the highly-selective region of the plan space is generally true but need not be always. In Figure 4.2, we see that even though we have only one plan when selectivities of both tables LINEITEM and NATION are below 20%, we have a number of clusters in this area. The *ideal* case would be to cluster the query-space based on the underlying plan diagram. Thus, we require *dynamic variable clustering*.

4.3 Improvements in Clustering

In this section, we introduce L^∞Norm as the new distance metric for clustering, followed by an algorithm that performs dynamic variable clustering. We then empirically show that dynamic variable clustering using L^∞Norm is better than previous approaches of clustering presented in Plastic.

4.3.1 Distance Function

Since L^1Norm is not adequate, we need a distance function that forms clusters that have a square shape or preferably a rectangle. L^∞Norm (also known as *Chessboard* distance metric) is one such metric. It clusters the entire query space into squares, the size of each square being determined by the threshold value. The new distance metric using L^∞Norm is shown below.

$$Distance(Q_1, Q_2) = \max_i(\frac{\mid ETS_1^i - ETS_2^i \mid}{max(TS_1^i, TS_2^i)}) \tag{4.3}$$

We can determine the *threshold value* for the distance function that needs to be set given the number of clusters[1].

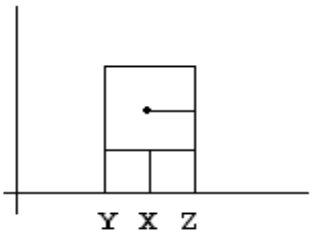


Figure 4.3: Square Shaped Cluster

Consider a single cluster as shown in Figure 4.3. Let δ be the threshold value. Let 'A' be the total size of the query-space that needs to be clustered. Let 'c' be the desired number of clusters.

The area of each cluster is then A/c. If 's' is the side of a square (shape of the cluster), then $s^2 = A/c$. Thus

[1]The number of clusters can be calculated from the space budget available for storing the clusters.

$$\delta^2 = (\frac{(Z-X)}{max(TS_1^i, TS_2^i)})^2 = \frac{s^2}{4K^2} = \frac{A}{4cK^2}$$

where K=$max(TS_1^i, TS_2^i)$ is the difference between the sizes of the two tables in the given query and hence does not change. Thus the required threshold is

$$\delta = \lceil \frac{1}{2K} * \sqrt{\frac{A}{c}} \rceil \tag{4.4}$$

If the cluster has n-dimensions, we can similarly show that the required threshold is

$$\delta = \lceil \frac{1}{2K} * \sqrt[n]{\frac{A}{c}} \rceil \tag{4.5}$$

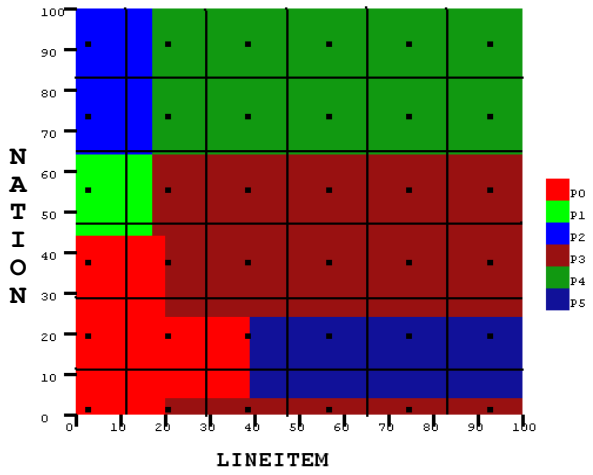


Figure 4.4: Fixed clustering with L^∞Norm

Our initial experiments have shown that using L^∞Norm as the distance metric itself increases the accuracy of clustering and makes it comparable to that of *static-variable* clustering using L^1Norm. For this particular query template Q-21 shown in Figure 4.1, we got an accuracy of 82.39% which is infact slightly more than the accuracy 81% we got for *static-variable* clustering. A sample output of fixed-sized clustering using L^∞Norm for *query template* Q-21 given in Figure 4.1 is shown in Figure 4.4.

4.3.2 Dynamic Variable Clustering

The algorithm for dynamic variable clustering is shown in Figure 4.5. This algorithm works on a fixed budget of clusters – given 'c' number of clusters, it merges and divides clusters in such a way that finally at most 'c' clusters are present. For each cluster, it identifies all the neighboring clusters that have same plan and merges them if the resulting cluster is in the form of a square.

```
INPUT  : C number of fixed size clusters
OUTPUT: K ≤ C number of dynamic sized clusters

DYNCLUSTER
{
      Let l be a bound on the size of the smallest cluster
      Let h be a bound on the size of the largest cluster
      Initial size of each fixed cluster s = ⌈√(l * h)⌉

      Get all eligible clusters 'E'.
      // A cluster is eligible if any 7 out of the 9 points shown in Figure 4.6
      // have the same plan.

1.    For each eligible cluster in E, gets its four cluster neighbors
2.    If all of them have same plan, cluster-size *= 4

3.    Repeat steps 1-4 with new cluster-size

4.    Sort clusters in decreasing order of their sizes

      // Initially removed[i]=false for all clusters
6.    For each eligible cluster in sorted list, if it is not removed
            Remove all clusters whose centers are within the boundary of this cluster

      Get the number of clusters removed = R
      // Since we only have a fixed budget,
      // clusters removed = clusters that can be added

7.    For each ineligible cluster, divide into four smaller clusters
            Decrement R by 4. if R ≤ 0 then break

}
```

Figure 4.5: Dynamic Clustering Algorithm

The initial size of each fixed cluster is taken as the geometric mean of the size of the maximum possible cluster to the size of the smallest allowed cluster. Then checking is done to determine the clusters eligible for merging. A cluster is eligible if it has the same plan for any 7 out of the 9 points shown in Figure 4.6. If a cluster is eligible and has four eligible neighbors with the same plan, then the size of the cluster is increased by a factor of 4. This is repeated for all the clusters.

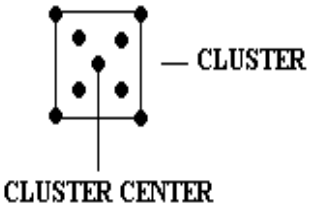


Figure 4.6: Points considered for eligibility

Next, starting from the cluster with the largest size, all clusters that have their centers within the boundary of this cluster are removed. This process is repeated for each cluster in decreasing order of their sizes, provided they are not already removed by clusters before them in the sorted order.

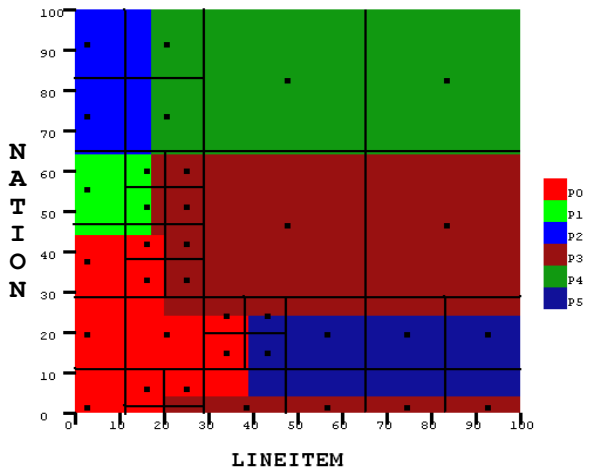


Figure 4.7: Dynamic Variable-Sized clustering

Finally the *ineligible clusters* are divided, generating four new clusters, subject to the condition that the number of new clusters generated can be atmost the number of clusters reduced in the previous step. A sample output of *dynamic-variable* clustering using L^{∞}Norm for *query template* Q-21 is shown in Figure 4.7.

Our initial experiments have confirmed our belief that *dynamic-variable* clustering increases the accuracy of clustering. For query template Q-21 shown in Figure 4.1, we obtained an accuracy of 89.95% with the same 'c' number of clusters which is substantially better than 81%, the accuracy obtained by using *static-variable* clustering.

4.4 Decision-Tree Classifier

In the original tool, identifying the matching cluster (if any) for the new query, was achieved by comparing the new query with the cluster representatives until either a similar representative was found, or all were found to be dissimilar. This process can become computationally expensive when a large number of clusters are present, as would often be the case.

Accordingly, we needed a faster technique such as decision tree or hierarchical clustering [3] – we have explored the former option since it naturally suits our problem. This is because most of our query features are deterministic as well as common to a small group of clusters and we can therefore have a set of comparisons that zero-in on the required cluster very quickly. For example, the feature set: *Degree Sequence* and *Join Predicate Index Counts*, will be the same for all the queries within the cluster and thus can be considered to be a characteristic of the cluster acting as a decision rule for selecting clusters. Another important advantage of decision trees is that once we have the rules generated, we can even drop the source query feature vectors and simply interpret clusters as leaves of the decision tree.

So, to hasten the process of cluster identification, we have now incorporated the popular C4.5 [43] decision-tree classifier which is present in the classifier module of the architecture diagram(see Figure 3.1). This new classifier module operates on the clusters in the database, after grouping them based on plan commonality – that is, clusters sharing the same plan are grouped together and a classifier is built on these groups.

To optimize the grouping process, initially the plan template of each query representative is traversed in post-order and an MD5 hash signature of this traversal is computed. Subsequently, these signatures, rather than the plans themselves, are compared to decide plan commonality among clusters. Such grouping significantly reduces the number of class labels in the cluster database, and has twofold advantages: Firstly, it increases the accuracy of the classifier, and secondly, results in a decision tree of lesser height, thereby requiring lesser time for classification. Quantitatively, our experiments show that the cluster identification time reduces by an *order of magnitude*, at only a small cost in the overall matching accuracy [22]. In fact, with the classifier, the identification cost is proportional to the *heterogeneity* of the clusters, whereas in the earlier version, the cost was proportional to the *cardinality* of the clusters. The decision tree also helps to identify the attributes of the query feature vector that have the most impact on plan choices.

We have preferred C4.5 over the newer version C5.0 since both have almost the same accuracy but the former is an open source and also works on both Windows and Linux Platforms. OC1 [36], another popular decision tree classifier which has oblique splits contrary to access parallel splits in C4.5, provides slightly better performance than C4.5 but unfortunately is not supported on the Windows Platform. A sample output of the classifier module is shown in Figure 4.8 (here, ETS refers to Effective Table Size, a query feature vector component [7] and there are three cluster groups).

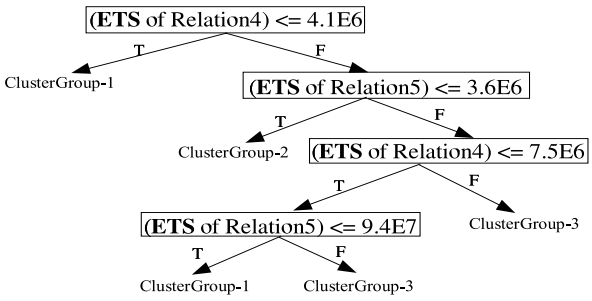


Figure 4.8: Classifier Module Output

4.5 Plan-Diff Module

The *plan-diff* module, intended for analyzing the specific differences between a pair of query execution plans, has been extended to compare plans across the four database platforms (DB2, Oracle, SQL-Server, PostgreSQL). This module identifies differences between plans using an adaption of the *X-Diff* [31] algorithm (which was proposed for computing differences between XML documents). This module now compares plans considering the *operator-level* attributes also. It can be used in two ways: (a) to compare plan choices for different versions of a query on a single platform, or (b) to compare choices for the same query across database platforms. Part of a sample output of the module, showing the differences between Plan 3 and Plan 4 of Figure 4.7, is shown in Figure 4.9 (the source plans are omitted because of their complexity and size). From this figure, we can see that these two plans differ only in the access method (index scan versus table scan) for the LINEITEM relation. We expect that this module will be especially beneficial for database administrators and query optimization researchers/students to help understand plan choices made by the optimizer.

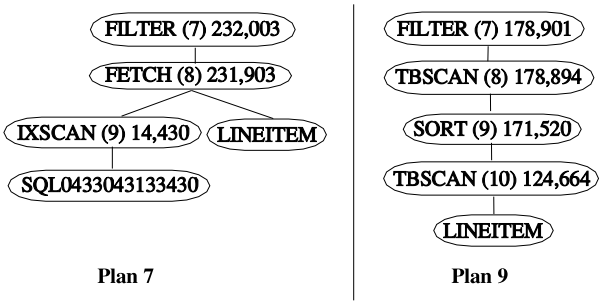


Figure 4.9: Plan Differences

Chapter 5

Picasso: Testbed Environment

In this section, we overview the Picasso tool and the experimental environment under which the plan and cost diagrams presented here were produced. We also show the integrated *plan-cost* and *plan-cardinality* diagrams that have been implemented in Picasso.

5.1 Picasso Tool

The Picasso tool is a value added tool that helps in understanding the intriguing behavior of modern optimizers. Through its GUI, users can submit a *query template* [7], the grid granularity at which instances of this template should be distributed across the plan space, the relations (axes) and their attributes on which the diagrams should be constructed, and the choice of query optimizer. A snapshot of the interface for a template based on Query 2 of the TPC-H benchmark, is shown in Figure 5.1 (the predicates "`p_size < C1`" and "`ps_supplycost < C2`" determine the selectivity axes).

With this information, the tool automatically generates SQL queries that are evenly spaced across the relational selectivity space (the statistics present in the database catalogs are used to compute the selectivities). For example, with a grid spacing of 100 x 100, a plan diagram is produced by firing 10000 queries, each query covering 0.01 percent of the plan diagram area. The resulting plans are stored persistently in the database, and in the post-processing phase, a unique color is assigned to each distinct plan, and the area covered by the plan is also estimated.

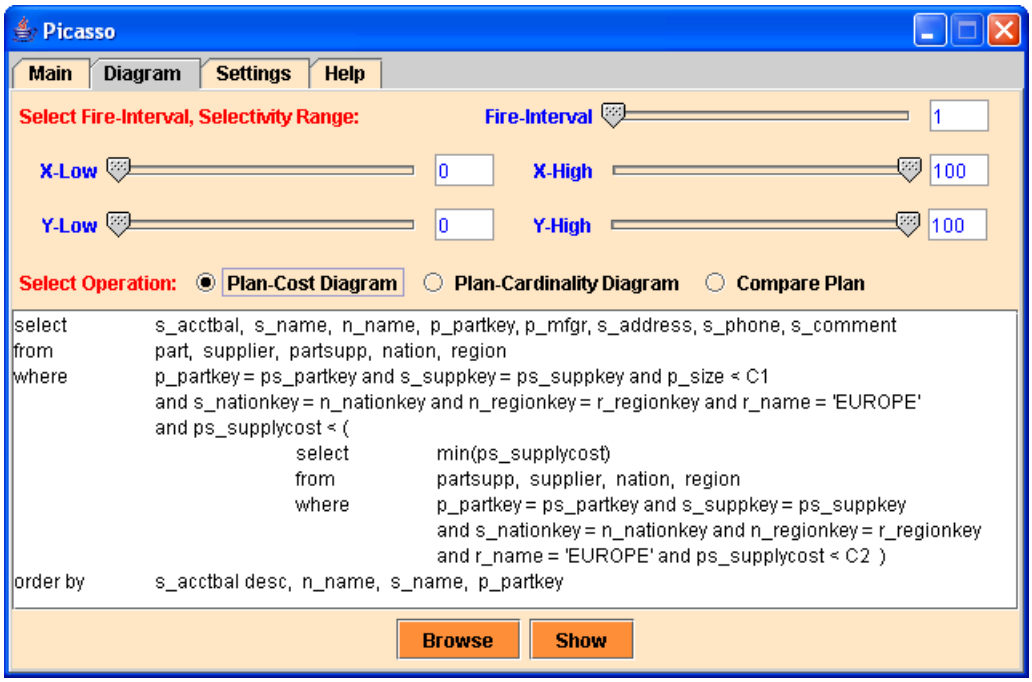


Figure 5.1: Picasso GUI

The space is then colored according to this assignment, and the legend shows (in ranked order) the space coverage of each plan. Differences between specific plans are easily identified using a *PlanDiff component* [22] that only requires dragging the cursor from one plan to the other in the plan diagram.

5.2 Database and Query Set

The database was created using the synthetic generator supplied with the TPC-H decision support benchmark, which represents a commercial manufacturing environment, featuring the following relations: REGION, NATION, SUPPLIER, CUSTOMER, PART, PARTSUPP, ORDERS and LINEITEM. A gigabyte-sized database was created on this schema, resulting in cardinalities of 5, 25, 10000, 150000, 200000, 800000, 1500000 and 6001215, for the respective relations.

All query templates were based on the TPC-H benchmark, which features a set of 22 queries, Q1 through Q22. To ensure coverage of the full range of selectivities, the relational axes in the plan diagrams are chosen only from the large-cardinality tables occurring in the query (i.e. NATION and REGION, which are very small, are not considered). An additional restriction

is that the selected tables should feature only in join predicates in the query, but not in any constant predicates. For a given choice of such tables, additional one-sided range predicates on attributes with high-cardinality domains in these tables are added to the queries to support a fine-grained continuous variation of the associated relational selectivities. As a case in point, the plan diagram in Figure 1.3(a) on the SUPPLIER and LINEITEM relations, was produced after adding to Q8 the predicates `s_acctbal` $<$ `C1` and `l_quantity` $<$ `C2`, where `C1` and `C2` are constants that are appropriately set to generate the desired selectivities on these relations. For making these changes we used ZQL [37], a SQL parser written in Java. In the remainder of this paper, for ease of exposition, we will use the benchmark query numbers for referring to the associated Picasso templates.

While plan and cost diagrams have been generated for most of the benchmark queries, we focus in the remainder of this paper only on those queries that have "dense" plan diagrams – that is, diagrams whose optimal plan set cardinality is 10 or more, making them interesting for analysis – for at least one of the commercial optimizers. For computational tractability, a query grid spacing of 100 x 100 is used, unless explicitly mentioned otherwise. Further, for ease of presentation and visualization, the query workloads are restricted to 2-dimensional selectivity spaces (with the exception of queries Q1 and Q6, which feature only a single relation, and therefore have a 1-D selectivity space by definition).

5.3 Relational Engines

The relational engines evaluated in our study are IBM DB2 v8.1 [39], Oracle 9i [41] and Microsoft SQL Server 2000 [40]. DB2 offers a range of optimization levels, going from 1 (lowest quality) through 5 (default) to 9 (highest quality). Oracle also has two optimization modes – *first_rows_n* and *all_rows* (default) – the former optimizes latency, while the latter optimizes query completion time.

While we have evaluated all the DB2 levels and all Oracle modes, for ease of exposition, the diagrams presented here, unless explicitly mentioned otherwise, were all obtained with the default of Level 5 for DB2, and *optimize-all-rows* for Oracle. An additional issue with regard to

DB2 is that inclusion of *hash-join* as a candidate join operator has to be explicitly set, and this was done. To support the making of informed plan choices, commands were issued to collect statistics on all the attributes participating in the queries. Finally, for every query submitted to the database systems, commands were issued to only "explain" the plan – that is, the plan to execute the query was generated, but not executed. This is because our focus here is on plan choices, and not on evaluating the accuracy of the associated cost estimations. As said earlier, due to legal restrictions, we randomly identify the three commercial optimizers as OptA, OptB and OptC, in the rest of the report.

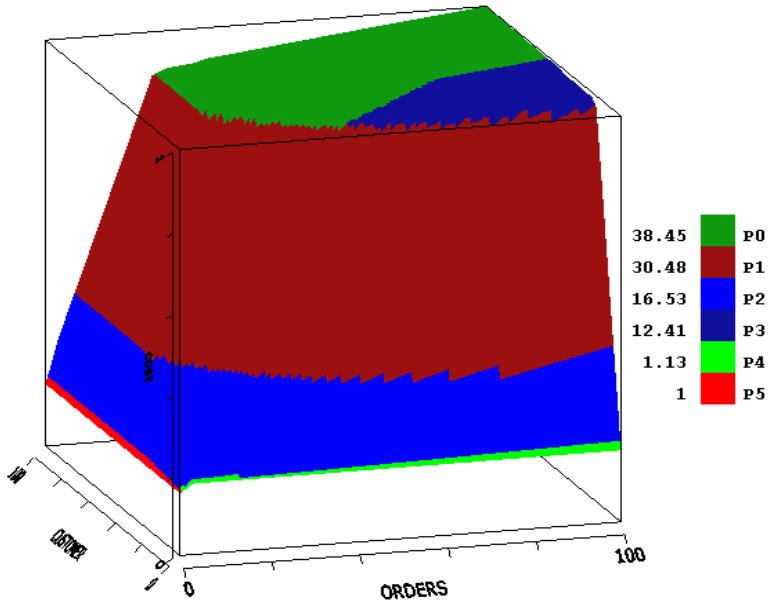
5.4 Computational Platform

A vanilla platform consisting of a Pentium-IV 2.4 GHz PC with 1 GB of main memory and 120 GB of hard disk, running the Windows XP Pro operating system, was used in our experiments. The three relational engines were all installed with their default configurations for all parameters, including those related to physical resources, except as mentioned above. For this platform, the complete set of evaluated queries and their associated plan, cost, and reduced-plan diagrams, over all three optimizers, are available at [32] – in the remainder of this report, we discuss their highlights.

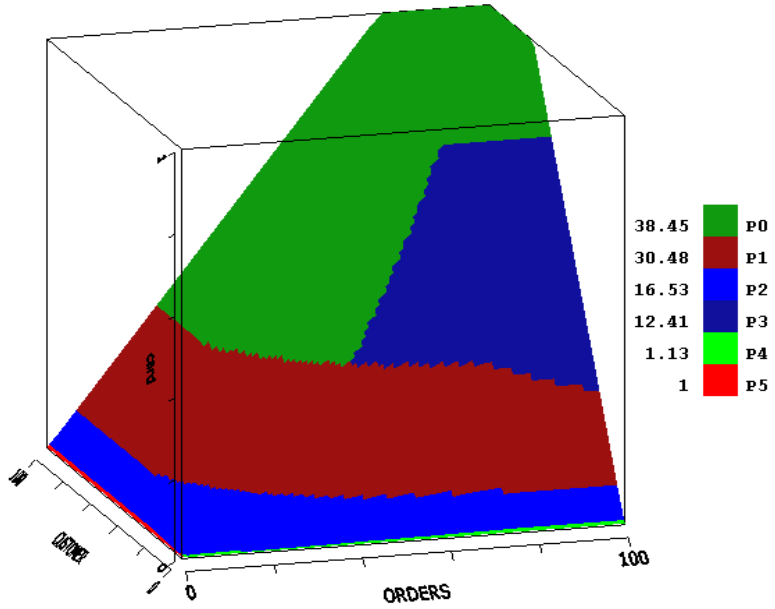
5.5 Integrated Plan-Cost/Plan-Cardinality Diagrams

An integrated *plan-cost* diagram is a 3D plan diagram with the Z-axis showing the normalized cost of each plan bounded in the interval [0, 1]. On similar lines, we define an integrated *plan-cardinality* diagram, as a 3D plan diagram which shows the normalized result cardinality on Z-axis. These diagrams are more useful when compared to an isolated plan diagram as they provide a more holistic view of the changes in plan space as a function of cost and result cardinalities. We have implemented a 3-D visualizer module, using VisAD [44] for generating these integrated plan-cost and plan-cardinality diagrams. VisAD, an acronym for "Visualization for algorithm development", has been developed actively by programmers in SSEC Visualization project at the university of Wisconsin Madison.

Sample 3D plan-cost and plan-cardinality diagrams for Query 7 of the TPC-H benchmark for the OptB optimizer are shown in Figure 5.2.



(a) **Integrated 3D Plan-Cost Diagram**



(b) **Integrated 3D Plan-Cardinality Diagram**

Figure 5.2: 3D Plan-Cost/Plan-Card Diagrams (Query 7, OptB)

Though plan-cost diagrams and plan-cardinality diagrams are more useful in understanding

the intricacies of optimizers, they are difficult to comprehend when presented as an image in a report. This can be seen by comparing the plan diagram shown in Figure 1.2(a) with the integrated plan-cost diagram shown in Figure 5.2(a). Hence we show only plan diagrams in the rest of the paper. Their corresponding cost diagrams shown in report are obtained by feeding the query points and their associated costs to a commercial 3-D visualizer – currently, the freeware Plot3D [38] is being used for this purpose.

Chapter 6

Skew in Plan Space Coverage

We start off our analysis of plan diagrams by investigating the *skew* in the space coverage of the optimal set of plans. In Table 6.1, we show for the various benchmark queries, three columns for each optimizer: First, the cardinality of the optimal plan set; second, the (minimum) percentage of plans required to cover 80 percent of the space; and, third, the Gini index [34], a popular measure of income inequality in economics – here we treat the space covered by each plan as its "income". Our choice of the Gini index is due to its desirable statistical properties including being Lorenz-consistent, and bounded on the closed interval [0,1], with 0 representing no skew and 1 representing extreme skew. Finally, the averages across all *dense queries* (10 or more plans in the plan diagram) are also given at the bottom of Table 6.1.

These statistics show that the cardinality of the optimal plan set can reach high values for a significant proportion of the queries. For example, the average (dense) cardinality is considerably in excess of *twenty*, across all three optimizers. Q9, in particular, results in more than 40 plans for all the optimizers. But it is also interesting to note that high plan density is not solely query-specific since there can be wide variations between the optimizers on individual queries – for example, Q18 results in 13 plans for OptB, but only 5 plans each for OptA and OptC. Conversely, OptB requires only 6 plans for Q7, but OptA and OptC employ 13 and 19 plans, respectively. It should also be noted that a common feature between Q8 and Q9, which both have large number of plans across all three systems, is that they are join-intensive nested queries with the outer query featuring *dynamic* base relations (i.e. the relations in the *from clause* are

TPC-H Query Number	OptA			OptB			OptC		
	Plan Card	80% Coverage	Gini Index	Plan Card	80% Coverage	Gini Index	Plan Card	80% Coverage	Gini Index
2	22	18%	0.76	14	21%	0.72	35	20%	0.77
5	21	19%	0.81	14	21%	0.74	18	17%	0.81
7	13	23%	0.73	6	50%	0.46	19	15%	0.79
8	31	16%	0.81	25	25%	0.72	38	18%	0.79
9	63	9%	0.88	44	27%	0.70	41	12%	0.83
10	24	16%	0.78	9	22%	0.69	8	25%	0.75
18	5	60%	0.33	13	38%	0.57	5	20%	0.75
21	27	22%	0.74	6	17%	0.80	22	18%	0.81
Avg(dense)	**28.7**	**17%**	**0.79**	**24.5**	**23%**	**0.72**	**28.8**	**16%**	**0.8**

Table 6.1: Skew in Plan Space Coverage

themselves the output of SQL queries).

When the fractional cardinality required to cover 80 percent of the space is considered, we see that on average it is in the neighborhood of 20 percent, highlighting the inequity in the plan space distribution. This is comprehensively captured by the Gini index values, which are mostly in excess of 0.5, and even reaching 0.8 on occasion, indicating very high skew in the plan space distribution. Further, note that this skew is present, *across the board, in all the optimizers*.

Overall, the statistics clearly demonstrate that modern optimizers tend to make extremely fine-grained choices. Further, these numbers are *conservative* in that they were obtained with a 100 x 100 grid – with finer-granularity grids, as mentioned in the Introduction, the number of plans often increased even further. For example, using a 1000 x 1000 grid for Q9 on OptB, the number of plans increased from 44 to 60!

6.1 Plan Cardinality Reduction by Swallowing

Motivated by the above skewed statistics, we now look into whether it is possible to replace many of the small-sized plans by larger-sized plans in the optimal plan set, without unduly increasing the cost of the query points associated with the small plans. That is, can small plans be "completely swallowed" by their larger siblings, leading to a reduced plan set cardinality, without materially affecting the associated queries.

To do this, we first fix a threshold, λ, representing the maximum percentage cost increase

that can be tolerated. Specifically, no query point in the original space should have its cost increased, *post-swallowing*, by more than λ. Next, to decide whether a plan can be swallowed, we use the following formulation:

Cost Domination Principle: Given a pair of distinct query points $q_1(x_1, y_1)$ and $q_2(x_2, y_2)$ in the two-dimensional selectivity space, we say that point q_2 dominates q_1, symbolized by $q_2 \succ q_1$, if and only if $x_2 \geq x_1$, $y_2 \geq y_1$, and result cardinality $R_{q_2} \geq R_{q_1}$ (note that result cardinality estimations are, in principle, independent of plan choices).[1] Then, if points $q_1(x_1, y_1)$ and $q_2(x_2, y_2)$, are associated with distinct plans P_1 and P_2, respectively, in the original space, C_1^2, the cost of executing query q_1 with plan P_2 is upper-bounded by C_2^2, the cost of executing q_2 with P_2, if and only if $q_2 \succ q_1$.

Intuitively, what is meant by the cost domination principle is that we expect the optimizer cost functions to be monotonically non-decreasing with increasing base relation selectivities and result cardinalities. Equivalently, a plan that processes a superset of the input, and produces a superset of the output, as compared to another plan, is estimated to be more costly to execute. However, as discussed later in Section 8, this (surprisingly) does not always prove to be the case with the current optimizers, and we therefore have to explicitly check for the applicability of the principle.

Based on the above principle, when considering swallowing possibilities for a query point q_s, we only look for replacements by "foreign" (i.e. belonging to a different plan) query points that are in the *first quadrant* relative to q_s as the origin, since these points upper-bound the cost of the plan at the origin. This is made clear in Figure 6.1, which shows that, independent of the cost model of the dominating plan, the cost of any foreign query point in the first quadrant will be an upper bound on the cost of executing the foreign plan at the swallowed point. We now need to find the set of dominating foreign points that are within the λ threshold, and if such points exist, choose one replacement from among these – currently, we choose the point with the lowest cost as the replacement. Finally, an *entire plan* can be swallowed if and only if *all* its query points can be swallowed by either a single plan or a group of plans. In our processing,

[1] Result cardinalities are usually monotonically non-decreasing with increasing x and y, but this need not always be the case, especially for nested queries.

we first order the plans in ascending order of size, and then go up the list, checking for the possibility of swallowing each plan.

Note that the cost domination principle is conservative in that it does not capture all swallowing possibilities, due to restricting its search only to the first quadrant. But, as we will show next, substantial reductions in plan space cardinalities can be achieved even with this conservative approach.

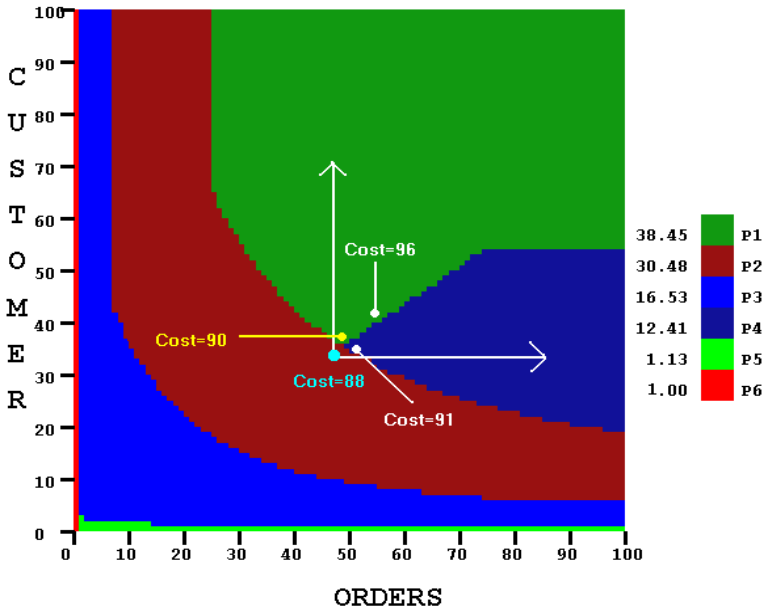


Figure 6.1: Dominating Quadrant

For the experiments presented here, we set λ, the cost increase threshold, to 10 percent. Note that in any case the cost computations made by query optimizers are themselves statistical *estimates*, and therefore allowing for a 10 percent "fudge factor" may be well within the bounds of the *inherent* error in the estimation process. In fact, as mentioned recently in [19, 28], cost estimates can often be signficantly off due to modeling errors, prompting the new wave of "learning" optimizers (e.g. LEO [28]) that iteratively refine their models to improve their estimates.

When the above plan-swallowing technique is implemented on the set of plans shown in Table 6.1, and with $\lambda = 10\%$, the resulting reductions (as a percentage) in the plan cardinalities are shown in Table 6.2. We see here that the reductions are very significant – for example, Q8 reduces by 87% (31 to 4), 84% (25 to 4) and 86% (38 to 5), for OptA, OptB and OptC,

TPC-H Query Number	OptA % Card Dec	OptA Avg Cost Inc	OptA Max Cost Inc	OptB % Card Dec	OptB Avg Cost Inc	OptB Max Cost Inc	OptC % Card Dec	OptC Avg Cost Inc	OptC Max Cost Inc
2	59.2	1.0	4.4	64.2	0.6	5.9	77.1	3.2	6.4
5	67.3	2.6	8.1	42.9	0.1	0.6	61.1	0.2	8.1
7	46.1	0.1	9.5	16.6	0.4	0.7	54.5	1.1	9.5
8	87.6	0.4	9.4	84	0.9	9.1	86.8	1.2	8.4
9	84.4	1.6	8.6	36.4	1.4	8.9	80.5	2.1	8.3
10	67.6	0.8	4.4	44.4	0.5	6.1	62.5	0.4	2.4
18	40.0	0.1	0.5	46.2	3.7	9.6	00.0	0.0	0.0
21	59.8	0.0	0.2	66.7	0.9	2.5	68.2	0.7	6.9
Avg(dense)	**67.4**	**0.9**	**6.4**	**56.9**	**0.7**	**6.1**	**71.4**	**1.4**	**7.9**

Table 6.2: Plan Cardinality Reduction by Swallowing

respectively. On average over dense queries, the reductions are of the order of 60% across all three optimizers, with OptC going over 70%. Also note that these reductions are *conservative* because when the grid granularity is increased – from 100 x 100 to, say, 1000 x 1000 – the new plans that emerge tend to be very small and are therefore highly likely to be subsequently swallowed. In a nutshell, the following thumb rule emerges from our results: *"two-thirds of the plans in a dense plan diagram are liable to be eliminated through plan swallowing"*.

In Table 6.2, we have also shown the *average* percentage increase in the costs of swallowed query points, as well as the *maximum* cost increase suffered across all query points. Note that, although the threshold is set to 10%, the *actual* average cost increase is rather low – less than 2%, which means that most of the swallowed query points *hardly* suffer on account of the replacement by an alternative plan. In fact, even the maximum increase does not always reach the threshold setting. Further, note that these averages and maxima are *upper bounds*, and the real cost estimates of the replacement plans at the swallowed points may be even lower in practice. Overall, our observation is that there appears to be significant potential to *drastically reduce the complexity of plan diagrams without materially affecting the query processing quality*.

A key implication of the above observation is the following: *Suppose it were possible to simplify current optimizers to produce only reduced plan diagrams, then the considerable computational overheads typically associated with the query optimization process may also be substantially lowered*. We suggest that this may be an interesting avenue to be explored by the

database research community.

6.2 Plan Reduction $\neq$ Optimization Levels

As mentioned earlier, optimizers typically have multiple optimization levels that trade off plan quality versus optimization time, and at first glance, our plan reduction technique may appear equivalent to choosing a coarser optimization level. However, the two concepts are completely different because the optimal plan sets chosen at different levels by the optimizer may be vastly dissimilar. A striking example is Q8, where *none* of the 68 plans chosen by OptA at the highest level are present among the 8 plans chosen at the lowest level. Further, going to a coarser level of optimization does not *necessarily* result in lower plan cardinalities – a case in point is OptA on Q2, producing only 4 plans at the highest level, but as many as 22 plans at a lower level. Again, there is zero overlap between the two optimal plan sets.

In contrast, with plan reduction by swallowing, only a subset of the *original plans* chosen by the optimizer are used to cover the entire plan space. In fact, plan reduction fits in perfectly with the query clustering approach previously proposed in our Plastic plan recycling tool [7, 23, 26, 33], where queries that are expected to have identical plan templates are grouped together based on similarities in their feature vectors. This is because the cluster regions *inherently* coarsen the plan diagram granularity.

Chapter 7

Relationship to PQO

A rich body of literature exists on *parametric query optimization* (PQO) [1, 2, 11, 12, 5, 6, 14, 16, 17]. The goal here is to apriori identify the optimal set of plans for the entire relational selectivity space at compile time, and subsequently to use at run time the actual selectivity parameter settings to identify the best plan – the expectation is that this would be much faster than optimizing the query from scratch. Most of this work is based on assuming cost functions that would result in one or more of the following:

Plan Convexity: If a plan P is optimal at point A and at point B, then it is optimal at all points on the line joining the two points;

Plan Uniqueness: An optimal plan P appears at only one contiguous region in the entire space;

Plan Homogeneity: An optimal plan P is optimal within the entire region enclosed by its plan boundaries.

However, we find that *none of the three* assumptions hold true, even approximately, in the plan diagrams produced by the commercial optimizers. For example, in Figure 1.3(a), plan convexity is severely violated by the regions covered by plans P12 (dark green) and P16 (light gray). The plan uniqueness property is violated by plan P4 (maroon) which appears in two non-contiguous locations in the top left quadrant, while plan P18 appears in finely-chopped pieces. Finally, plan homogeneity is violated by the small (brown) rectangle of plan P14, close to coordinates (60,30), in the (light-pink) optimality region of plan P3.

The prior literature [12, 17] had also estimated that *high plan densities* are to be expected only along the selectivity axes – that is, where one or both base relations in the plan diagram are extremely selective, providing only a few tuples. However, we have found that high plan densities can be present elsewhere in the selectivity space also – for example, see the region between plans P5 (dark brown) and P11 (orange) in Figure 1.3(a). This is also the reason for our choosing a uniform distribution of the query instances, instead of the exponential distribution towards lower selectivity values used in [12].

In the following section, more detailed statistics about the violations of the above assumptions are presented, as part of a discussion on interesting plan diagram patterns.

Chapter 8

Interesting Plan Diagram Patterns

We now move on to presenting representative instances of a variety of interesting patterns that emerged in the plan diagrams across the various queries and optimizers that we evaluated in our study.

8.1 Plan Duplicates and Plan Islands

In several plan diagrams, we noticed that a given optimal plan may have *duplicates* in that it may appear in several different disjoint locations. Further, these duplicates may also be spatially quite separated. For example, consider the plan diagram for Q10 with OptA in Figure 8.1. Here, we see that plan P3 (dark pink) is present twice, being present both in the center, as well as along the right boundary of the plan space. An even more extreme example is plan P6 (dark green), which is present around the 20% and 95% markers on the CUSTOMER selectivity axis.

A different kind of duplicate pattern is seen for Q5 with OptC, shown in Figure 8.2, where plan P7 (magenta) is present in three different locations, all within the confines of the region occupied by plan P1 (dark orange). When plans P7 and P1 are compared, we find that the former uses a nested-loops join between the small relations NATION and REGION, whereas the latter employs a sort-merge-join instead.

Apart from duplicates, we also see that there are instances of *plan islands*, where a plan region is completely enclosed by another. For example, plan P18 is a (magenta) island in the

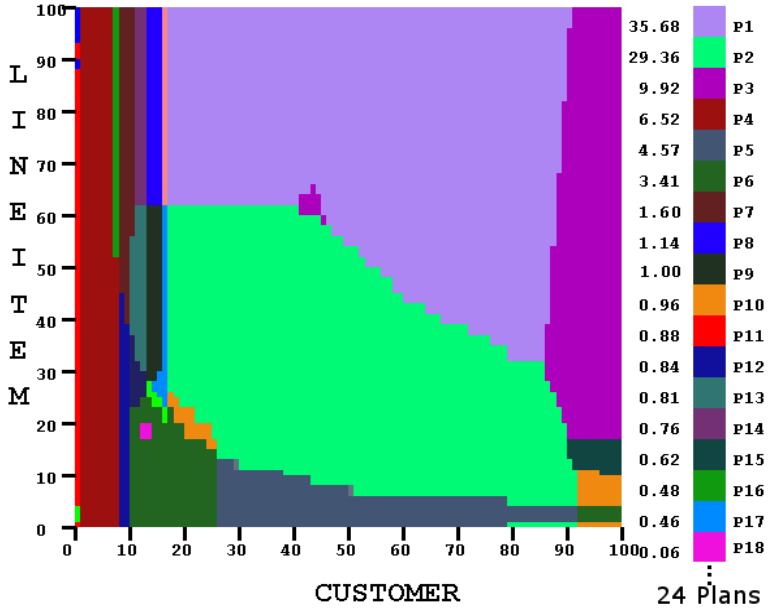


Figure 8.1: Duplicates and Islands (Query 10, OptA)

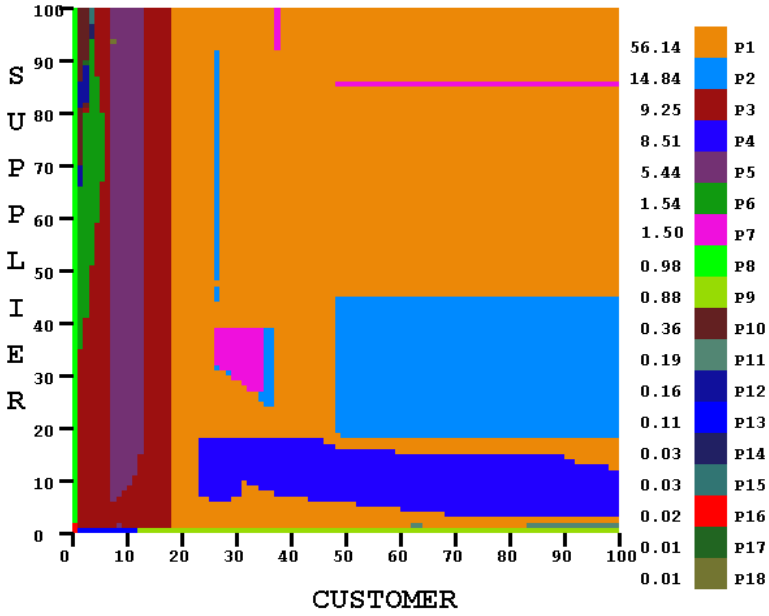


Figure 8.2: Duplicates and Islands (Query 5, OptC)

optimality region of the (dark green) plan P6 in the lower left quadrant of Figure 8.1. Investigating the internals of these plans, we find that plan P18 has a hash-join between CUSTOMER and NATION followed by a hash-join with a sub-tree whose root is a nested-loop join. The only difference in plan P6 is that it first hash-joins the CUSTOMER relation with the sub-tree,

and then performs the hash-join with NATION.

The number of such duplicates and islands for each optimizer, over all dense queries of the benchmark, is presented in Table 8.1 (Original columns). We see here that all three optimizers generate a significant number of duplicates; OptA also generates a large number of islands, whereas OptB and OptC have relatively few islands.

Databases	**# Duplicates**		**# Islands**	
	Original	**Reduced**	**Original**	**Reduced**
OptA	130	13	38	3
OptB	80	15	1	0
OptC	55	7	8	3

Table 8.1: Duplicates and Islands

In general, the reason for the occurrence of such duplicates and islands is that two or more competing plans have fairly close costs in that area. So, the optimizer due to its extremely fine grained plan choices, obtains plan diagrams with these features. This is confirmed from Table 8.1 (Reduced columns), where after application of the plan reduction algorithm, a significant decrease is observed in the number of islands and duplicates. This also means that PQO, which, as mentioned in the previous section, appears ill-suited to directly capture the complexities of modern optimizers, may turn out to be a more viable proposition in the space of reduced plan diagrams.

8.2 Plan Switch Points

In several plan diagrams, we find lines parallel to the axes that run through the *entire* selectivity space, with a plan shift occurring for *all* plans bordering the line, when we move across the line. We will hereafter refer to such lines as “plan switch-points”.

In the plan diagram of Figure 8.3, obtained with Q9 on OptA, an example switch-point appears at approximately 30% selectivity of the SUPPLIER relation. Here, we found a *common change* in all plans across the switch-point – the hash-join sequence PARTSUPP $\bowtie$ SUPPLIER $\bowtie$ PART is altered to PARTSUPP $\bowtie$ PART $\bowtie$ SUPPLIER, suggesting an intersection of the cost

function of the two sequences at this switch-point.

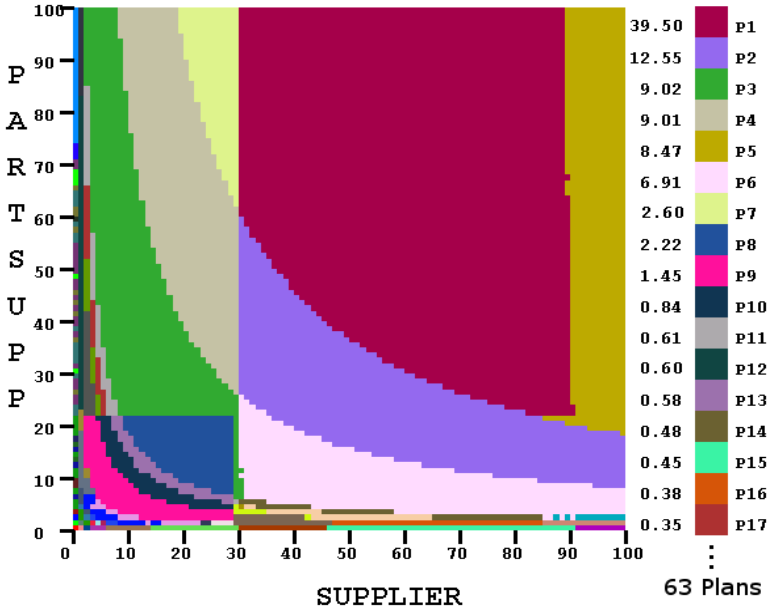


Figure 8.3: Plan Switch-Point (Query 9, OptA)

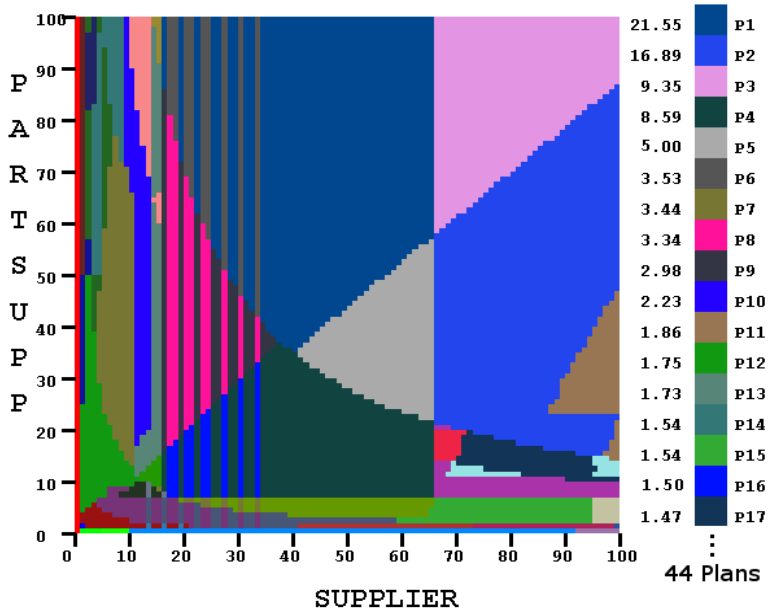


Figure 8.4: Venetian Blinds Pattern (Query 9, OptB)

For the same Q9 query, an even more interesting switch-point example is obtained with OptB, shown in Figure 8.4. Here we observe, between 10% and 35% on the SUPPLIER axis, *six plans* simultaneously changing with rapid alternations to produce a “Venetian blinds” effect.

Specifically, the optimizer changes from P6 to P1, P16 to P4, P25 to P23, P7 to P18, P8 to P9, and P42 to P47, from one vertical strip to the next.

The reason for this behavior is that the optimizer alternates between a *left-deep* hash join and a *right-deep* hash join across the NATION, SUPPLIER and LINEITEM relations. Both variations have almost equal estimated cost, and their cost-models are perhaps discretized in a step-function manner, resulting in the observed blinds.

8.3 Speckle Pattern

Operating Picasso with Q17 on OptA results in Figure 8.5. We see here that the entire plan diagram is divided into just two plans, P1 and P2, occupying nearly equal areas, but that plan P1 (bright green) also appears as speckles sprinkled in P2's (red) area.

The only difference between the two plans is that an additional SORT operation is present in P1 on the PART relation. However, the cost of this sort is very low, and therefore we find intermixing of plans due to the close and perhaps discretized cost models.

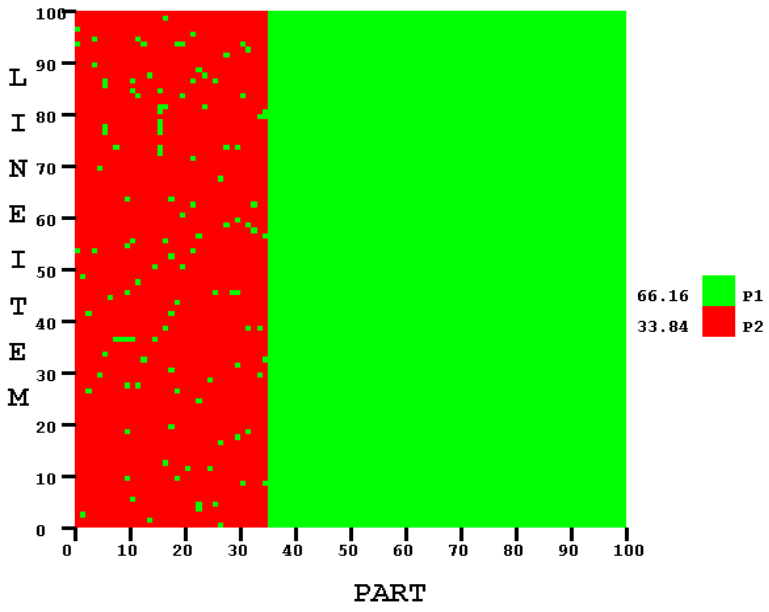
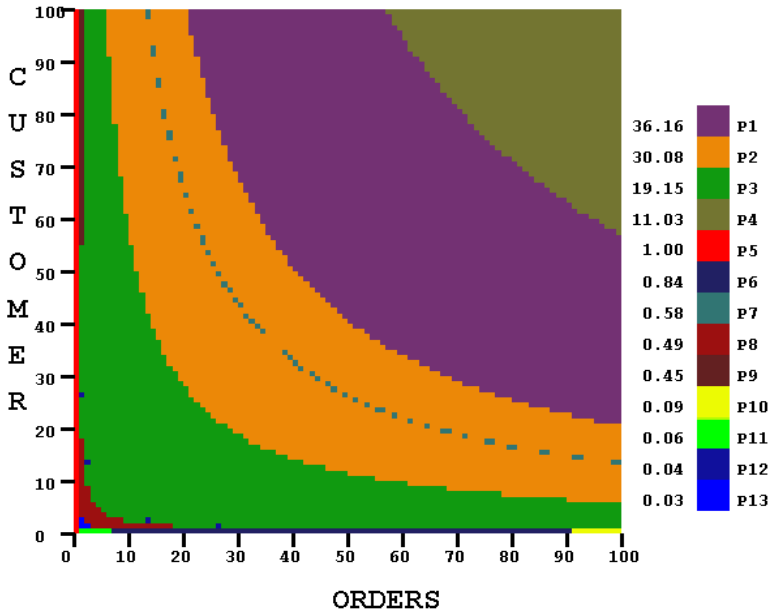
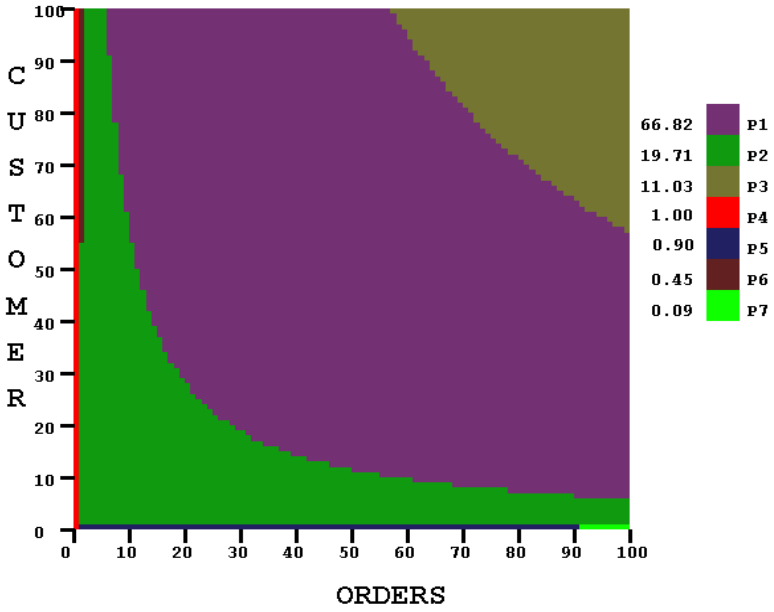


Figure 8.5: Speckle Pattern (Query 17, OptA)

8.4 Footprint Pattern



(a) **Footprint Pattern**



(b) **Reduced Footprint Pattern**

Figure 8.6: Plan Diagram and Reduced Plan Diagram (Query 7, OptA)

A curious pattern, similar to footprints on the beach, shows up in Figure 8.6(a), obtained

with Q7 on the OptA optimizer, where we see plan P7 exhibiting a thin (cadet-blue) broken curved pattern in the middle of plan P2's (orange) region. The reason for this behavior is that both plans are of roughly equal cost, with the difference being that in plan P2, the SUPPLIER relation participates in a sort-merge-join at the top of the plan tree, whereas in P7, the hash-join operator is used instead at the same location. This is confirmed in the corresponding reduced plan diagram e 8.6(b) where the footprints disappear.

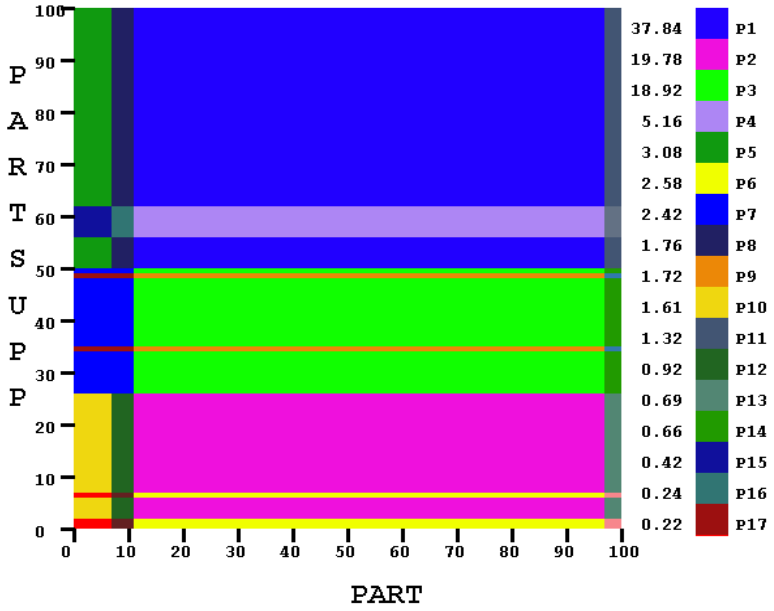
8.5 Non-Monotonic Cost Behavior

The example switch-points shown earlier, were all *cost-based* switch-points, where plans were switched to derive lower execution costs. Yet another example of such a switch-point is seen in Figure 8.7(a), obtained with query Q2 on OptA, at 97% selectivity of the PART relation. Here, the common change in all plans across the switch-point is that the hash-join between relations PART and PARTSUPP is replaced by a sort-merge-join.

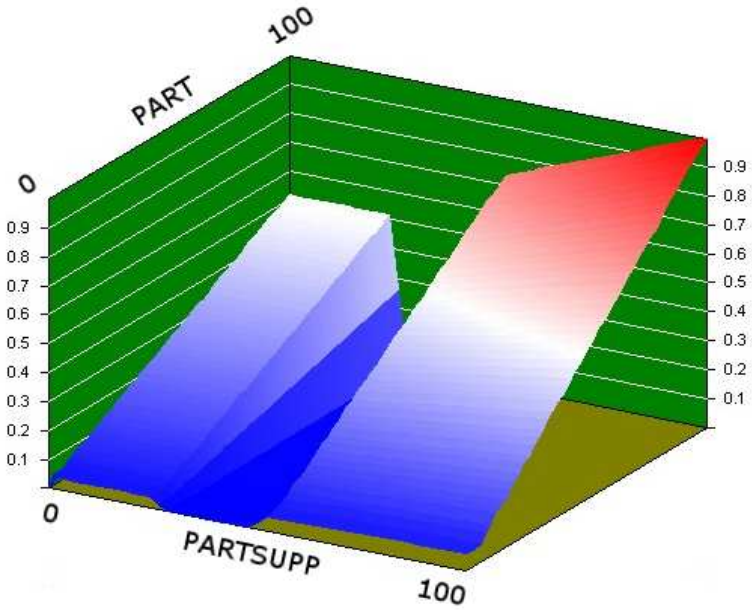
But, in the same picture, there are switch-points occurring at 26% and 50% in the PARTSUPP selectivity range, that result in a counter-intuitive *non-monotonic* cost behavior, as shown in the corresponding cost diagram of Figure 8.7(b). Here, we see that although the result cardinalities are monotonically increasing, the estimated costs for producing these results show a marked non-monotonic step-down behavior in the middle section. Specifically, at the 26% switch-point, an additional 'sort' operator (on `ps_supplycost`) is added, which substantially decreases the overall cost – for example, in moving from plan P2 to P3 at 50% PART selectivity, the estimated cost decreases by a factor of 50! Conversely, in moving from P3 to P1 at the 50% switch-point, the cost of the optimal plan jumps up by a factor of 70 at 50% PART selectivity.

Step-function upward jumps in the cost with increasing input cardinalities are known to occur – for example, when one of the relations in a join ceases to fit completely within the available memory – however, what is surprising in the above is the step-function cost *decrease* at the 26% switch-point. We conjecture that such disruptive cost behavior may arise either due to the presence of rules in the optimizer, or due to parameterized changes in the search space evaluated by the optimizer.

The above example showed non-monotonic behavior arising out of a plan-switch. However, more surprisingly, we have also encountered situations where a plan shows non-monotonic behavior *internal* to its optimality region. A specific example is shown in Figure 8.8 obtained for Q21 with OptA. Here, the plans P1, P3, P4 and P6, show a reduction in their estimated costs with increasing input and result cardinalities. An investigation of these plans showed that all of them feature a nested-loops join, whose estimated cost *decreases* with increasing cardinalities of its input relations – this may perhaps indicate an inconsistency in the associated cost model. Further, such instances of non-monotonic behavior were observed with all three optimizers.

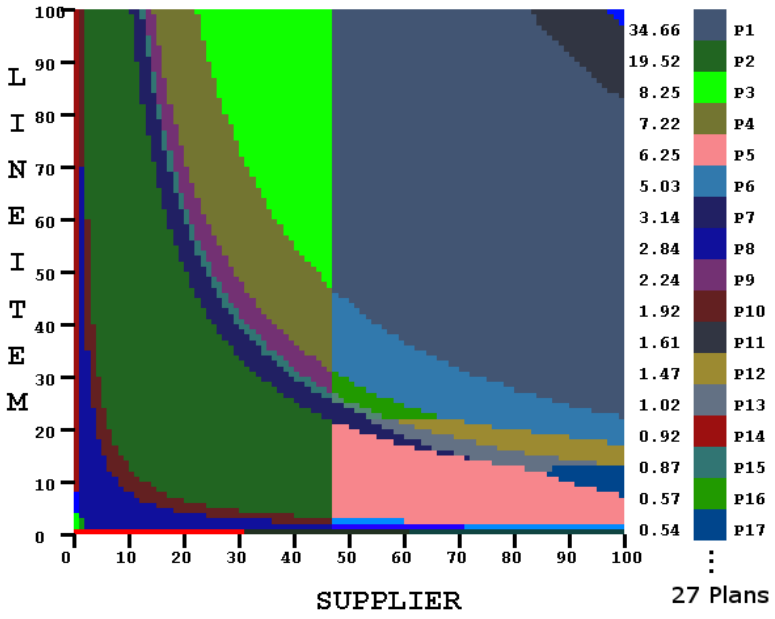


(a) **Plan Diagram**

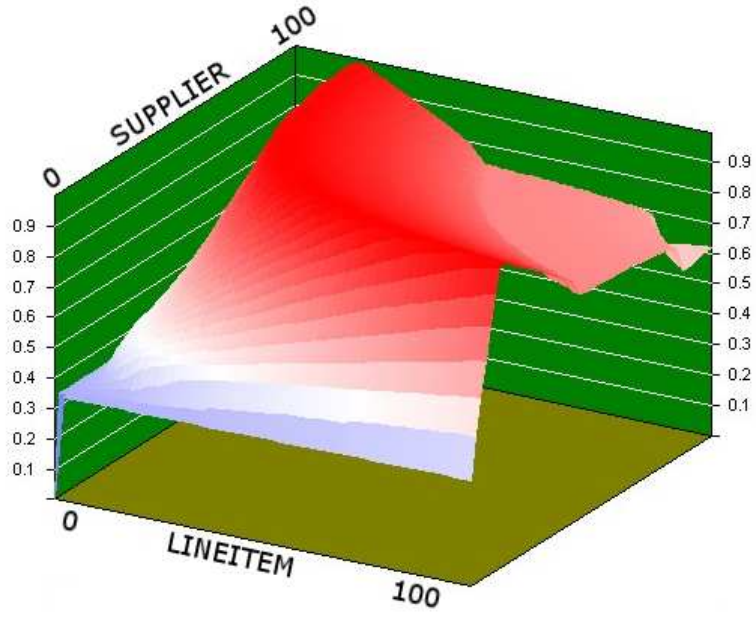


(b) **Cost Diagram**

Figure 8.7: Plan-Switch Non-Monotonic Costs (Query 2, OptA)



(a) **Plan Diagram**



(b) **Cost Diagram**

Figure 8.8: Intra-plan Non-Monotonic Costs (Query 21, OptA)

Chapter 9

Conclusions and Future Work

In this report, we have attempted to improve both the speed and the quality of results of the optimizer. Plastic, a tool that significantly improves the speed of the optimizer, has been improved by incorporating a host of new features into it.

Using L^{∞}*Norm* as the distance metric, we have proposed and implemented *dynamic variable-sized* that can adjust gracefully to both high-volatility and low-volatility regions of plan space. Through query template Q-21 shown in Figure 4.1, we have demonstrated that *dynamic-variable* clustering improves classification accuracy when compared to *static-variable* clustering. We have incorporated C4.5 decision tree into Plastic for fast cluster identification. We first group the clusters based on plan commonality and then build a classifier on these groups of clusters. We have extended the *plan-diff* module, which can now be used to compare plan choices across database platforms, along with comparing plan choices for different versions of same query. This plan comparison is currently performed taking into consideration operator-level attributes of the plans.

We have conceived and developed a tool called Picasso that helps in gaining useful insights in understanding the intriguing behavior of modern optimizers. Using Picasso, we have attempted to analyze the behavior of (1-D and 2-D) plan and cost diagrams produced by modern optimizers on queries based on the TPC-H benchmark. Our study shows that many of the queries result in highly intricate diagrams, with several tens of plans covering the space. Further, there is heavy skew in the relative coverage of the plans, with 80 percent of the space typically

covered by 20 percent or less of the plans. We showed that through a process of plan reduction where the query points associated with a small-sized plan are swallowed by a larger plan, it is possible to significantly bring down the cardinality of the plan diagram, without materially affecting the query cost.

We also demonstrated that a variety of complex and intricate patterns are produced in the diagrams, which may be an overkill given the coarseness of the underlying cost space. These patterns also indicate that the basic assumptions of parametric query optimization literature do not hold in practice. However, with reduced plan diagrams, the gap between theory and practice is considerably narrowed.

Not being privy to the internals of optimizers, our work is perforce speculative in nature. However, we hope that it may serve as a stimulus to the database research community to investigate mechanisms for pruning the plan search space so as to directly generate reduced plan diagrams, and thereby perhaps achieve substantial savings in the significant overheads normally associated with the query optimization process.

In future, we plan to extend Plastic, which currently only handles basic SPJ queries, to support nested queries, groups and aggregates. We would also like to conduct a deeper investigation into the kinds of queries that result in dense plan diagrams, such as, for example, the presence of dynamic base relations. Also, a major limitation of our current work is its restriction to 1-D and 2-D plan diagrams – in practice, there may be many more schema and system dimensions affecting plan choices. Therefore, we intend to investigate higher dimensional plan diagrams in our future research.

Chapter 10

Appendix

Towards the end of the project, we have also evaluated the PostgreSQL 8.0.3 [42] relational engine. This optimizer [4] has two optimization levels, one which uses *Genetic Query Optimizer*, and the other that does exhaustive search. We could do only a preliminary study for PostgreSQL. We hope to do a more detailed study of the plan diagrams generated in PostgreSQL as part of our future work. We present here the results of our preliminary experiments for the optimization level that uses Genetic Query Optimizer here. The complete set of evaluated queries and their associated plan, cost, and reduced-plan diagrams for PostgreSQL is available at [32].

TPC-H Query Number	**PostgreSQL**		
	Plan Cardinality	**80% Coverage**	**Gini Index**
2	7	28%	0.75
5	15	20%	0.80
7	19	21%	0.77
8	23	13%	0.82
9	9	22%	0.78
10	11	27%	0.72
18	19	31%	0.70
21	25	20%	0.79
Avg(dense)	**18.6**	**22%**	**0.77**

Table 10.1: Skew in Plan Space Coverage for PostgreSQL

TPC-H Query Number	PostgreSQL % Cardinality Decrease	PostgreSQL Avgerage Cost Increase	PostgreSQL Maximum Cost Increase
2	28.5	6.2	6.4
5	33.3	1.5	6.1
7	68.4	2.0	9.2
8	82.6	1.1	4.3
9	77.7	0.5	2.6
10	36.4	1.2	2.6
18	47.4	1.9	9.9
21	80.0	0.0	0.0
Avg(dense)	**58.0**	**1.3**	**5.4**

Table 10.2: Plan Cardinality Reduction by Swallowing in PostgreSQL

Databases	# Duplicates		# Islands	
	Original	Reduced	Original	Reduced
PostgreSQL	59	4	10	0

Table 10.3: Duplicates and Islands in PostgreSQL

Bibliography

[1] A. Betawadkar, "Query Optimization with One Parameter", *Master's Thesis, Dept. of Computer Science and Engineering, IIT Kanpur*, February 1999.

[2] R. Cole and G. Graefe, "Optimization of dynamic query evaluation plans", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1994.

[3] R. Duda, P. Hart and D. Stork, "Pattern Classification", *Wiley-Interscience, 2nd edition*, 2000.

[4] Z. Fong, "The design and implementation of the Postgres query optimizer" *Masters thesis, Dept. of Computer Science, University of California, Berkeley*, April 1986.

[5] S. Ganguly, "Design and analysis of Parametric Query Optimization Algorithms", *Proc. of 24th Intl. Conf. on Very Large Data Bases*, August 1998.

[6] S. Ganguly and R. Krishnamurthy, "Parametric Query Optimization for distributed databases based on load conditions", *Proc. of COMAD Intl. Conf. on Management of Data*, December 1994.

[7] 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*, August 2002.

[8] R. Gopal and R. Ramesh, "The Query Clustering Problem: A Set Partitioning Approach", *IEEE Trans. on Knowledge and Data Engineering*, 7(6), December 1995.

[9] G. Graefe and W. McKenna, "The Volcano optimizer generator: Extensibility and efficient search", *Proc. of 9th IEEE Intl. Conf. on Data Engineering (ICDE)*, April 1993.

[10] A. Hartigan, "Clustering Algorithms", *John Wiley and Sons, New York*, 1975.

[11] 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*, August 2002.

[12] A. Hulgeri and S. Sudarshan, "AniPQO: Almost Non-intrusive Parametric Query Optimization for Nonlinear Cost Functions", *Proc. of 29th Intl. Conf. on Very Large Data Bases*, September 2003.

[13] I. Ilyas, J. Rao, G. Lohman, D. Gao and E. Lin, "Estimating Compilation Time of a Query Optimizer", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2003.

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

[15] J. Park and A. Segev, "Using Common Subexpressions to Optimize Multiple Queries", *Proc. of 4th IEEE Intl. Conf. on Data Engineering* , February 1988.

[16] V. Prasad, "Parametric Query Optimization: A Geometric Approach", *Master's Thesis, Dept. of Computer Science and Engineering, IIT Kanpur*, April 1999.

[17] S. Rao, "Parametric Query Optimization: A Non-Geometric Approach", *Master's Thesis, Dept. of Computer Science and Engineering, IIT Kanpur*, March 1999.

[18] N. Reddy and J. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers", *Proc. of 31st Intl. Conf. on Very Large Data Bases*, September 2005.

[19] 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.

[20] A. Rosenthal, U. Dayal and D. Reiner, "Speeding a query optimizer: the pilot pass approach", *Unpublished Manuscript*, Computer Corporation of America, 1990.

[21] 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*, June 2000.

[22] P. Sarda, "Green Query Optimization: Taming Query Optimization Overheads through Plan Recycling", *Master's Thesis, Dept. of Computer Science and Automation, IISc Bangalore*, May 2004.

[23] P. Sarda and J. Haritsa, "Green Query Optimization: Taming Query Optimization Overheads through Plan Recycling", *Proc. of 30th Intl. Conf. on Very Large Data Bases*, September 2004.

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

[25] T. Sellis, "Multiple Query Optimization", *ACM Trans. on Database Systems*, 13(1), March 1988.

[26] V. Sengar and J. Haritsa, "PLASTIC: Reducing Query Optimization Overheads through Plan Recycling", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2003.

[27] K. Shim, T. Sellis and D. Nau, "Improvements on a heuristic algorithm for multiple-query optimization", *Data and Knowledge Engineering*, 12, February 1994.

[28] M. Stillger, G. Lohman, V. Markl and M. Kandil, "LEO – DB2's LEarning Optimizer", *Proc. of 27th VLDB Intl. Conf. on Very Large Data Bases (VLDB)*, September 2001.

[29] M. Stonebraker, J. Frew, K. Gardels and J. Meredith, "The SEQUOIA 2000 Storage Benchmark", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1993.

[30] 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.

[31] Y. Wang, D. DeWitt and J.Cai, "X-Diff: A Fast Change Detection Algorithm for XML-Document", *Proc. of 19th IEEE Intl. Conf. on Data Engineering*, March 2003.

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

[33] *http://dsl.serc.iisc.ernet.in/projects/PLASTIC*

[34] *http://en.wikipedia.org/wiki/Gini_coefficient*

[35] *http://www.artlex.com/ArtLex/c/cubism.html*

[36] *http://www.cs.jhu.edu/ salzberg/announce-oc1.html*

[37] *http://www.experlog.com/gibello/zql*

[38] *http://www.orchardhouse.vtrading.co.uk/Plot3D.htm*

[39] *http://www-306.ibm.com/software/data/db2/udb/v8/*

[40] *http://www.microsoft.com/sql/techinfo/productdoc/ 2000/books.asp*

[41] *http://www.oracle.com/technology/products/ oracle9i/index.html*

[42] *http://www.postgresql.org/download/*

[43] *http://sourceforge.net/projects/weka*

[44] *http://www.ssec.wisc.edu/~billh/visad.html*

[45] *http://www.tpc.org/tpch*