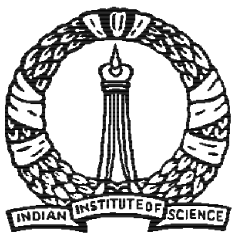
# Picasso: Analyzing and Characterizing Relational Query Optimizers

A Project Report
Submitted in Partial Fulfilment of the
Requirements for the Degree of
**Master of Engineering**
in
Internet Science and Engineering

by

**Akshat Nair**




Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012

July 2006

***To***

***My Family***

*for supporting and encouraging me in all my endeavors.*

# Acknowledgments

First of all I would like to thank my guide Prof. Jayant R. Haritsa for his encouragement and support. I had a very good experience under his supervision and I learnt lot of things from him.

I am deeply indebted to my sisters who have been a constant source of support and inspiration for me. I would also like to thank my fellow DSL'ites Aslam, Gopal, Sandeep and Prateem for their cheerful and helpful company which made working much more easier in the lab. Last, but not least, my friends and relatives who have in some or the other way helped me in completing this work deserve thanks.

# Abstract

Query optimizers are an integral component of widely used relational database management systems. The performance of these systems is highly affected by the efficiency of query optimizers. Picasso, a tool proposed in [7], has elucidated some problems in the existing relational query optimizers in a graphical way. A *plan diagram* [2] is a pictorial enumeration of the execution plan choices of a database query optimizer over the relational selectivity space. Using Picasso, an analysis of plan diagrams on a suite of industrial strength query optimizers for queries based on the *TPC-H benchmark* [24] can be found in [7]. These diagrams provide a variety of striking and interesting revelations, including that current optimizers make extremely fine-grained plan choices; that the plan optimality regions may have highly intricate patterns and irregular boundaries.

Picasso was available on three industrial strength query optimizers, DB2, Oracle and MS SQL Server. We have now ported it on two more query optimizers: Sybase and PostgreSQL. This has completed the spectrum of commercially and widely used databases. We have done an analysis of *plan-cardinality diagrams* [7] that has revealed that the optimizer's estimate of cardinality ar off by huge margin. The number of plans that optimizer chooses was observed to increase with the query grid resolution [7]. Another interesting observation was that*cost increase threshold* [7] value of 10% produced most of the reduction possible for a plan diagram.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

SQL the standard database query language is declarative in nature. It tells the database system *what to do* but does not specify *how to do*. For example consider the query shown in Figure 1 which fetches the name of employee and his depart name who earn less than 8000, it does not specify how to join the tables *emp* and *dept*, whether to use *Nested-Loops join, Sort-Merge join or Hash join*. Also the join order has been left unspecified.

```
select
    empName, deptName
from
    emp, dept
where
    emp.deptNo = dept.deptNo
    and salary < 8000
```

Figure 1.1: Example Query

In a database system it is the responsibility of module called *Query optimizer* to come up with strategies, known as execution plans, for executing the given SQL query. But this process of query optimization is computationally intensive and hence makes query optimizer a vital component of modern database systems. The performance of the system is highly dependent on the performance and functionality of the optimizer. The difference between the cost of best

execution plan and any other randomly chosen plan could be in orders of magnitude. Hence query optimization is a necessary step.

Most of the current query optimizers are cost based, with few rules included. The optimizer takes the given query and with the help of system catalogs and cost model comes up with a minimum cost plan. The efficiency of plan is measured in terms of query response time. Coming up with an efficient plan is important as the difference between the cost of the best execution plan and a sub-optimal plan could be enormous. An efficient optimizer should come up with a good execution plan without consuming too much time. As the complexity of query increases the time required to optimize it also increases. The need for an efficient optimizer is indubitably evident from the complex queries of *TPC-H decision support benchmark* [24].

The optimal plan choice given by a cost based query optimizer is mainly a function of the *selectivities* of the relations participating in the query, for a given database and system configuration. *Selectivity* is the estimated number of relevant tuples or rows of a relation that are relevant for producing the result of the query.

### 1.0.1 The Picasso Tool

Recently a tool called Picasso was introduced in [7]. Picasso is a JAVA tool which when supplied with a query template and a relational engine, automatically generates the associated *plan, plan-cost and plan-cardinality diagrams*.

A "plan diagram" is a color-coded pictorial enumeration of the execution plan choices of a database query optimizer over the relational selectivity space. For a query shown in Figure 1.0.1, based on Query 3 of the TPC-H benchmark, with selectivity variations on the ORDERS and LINEITEM relations an example plan diagram, selected by the OptB optimizer, is shown in Figure 1.3(a).

The diagram displays 6 optimal plans over the selectivity space. The area covered by each plan in this region has been indicated next to the plan number in the legend.

Complementary to the plan diagram is a "plan-cost diagram", shown in Figure 1.3(b), which is a three-dimensional visualization of the estimated plan execution costs over the same relational selectivity space as the plan diagram. The costs are normalized to the maximum cost over

```
select
    l_orderkey,
    sum(l_extendedprice *
    (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_totalprice    <=  :1
    and l_extendedprice <= :2
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
```

Figure 1.2: Query Template

the space, and the diagram has been drawn using the plan color encoding.

Query optimizers apart from estimating the cost of executing the query also try to estimate the result cardinality. Cardinality is defined as the number of tuples that comprise the result. Providing a third dimension to the analysis of query optimizer, Picasso generates "plan-cardinality diagram" which is a three-dimensional visualization of the estimated result cardinalities over the same relational selectivity space as the plan diagram. Figure 1.4(a) shows the plan-cardinality diagram. The plan-cardinality diagrams are also encoded using plan color encoding.

### 1.0.2 Plan Diagram Cardinality Reduction by Swallowing

Picasso demonstrated that 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, small plans can be "completely swallowed" by their larger siblings, leading to a reduced plan set cardinality, without materially affecting the associated cost.

The experiments in [7] were done by setting $\lambda$, the cost increase threshold, to 10 percent. Very significant reductions in plan cardinality were obtained in the process. On average over dense queries, the reductions were of the order of 60% across all three optimizers. The reduced plan diagram for example query is shown in Figure 1.4(b).

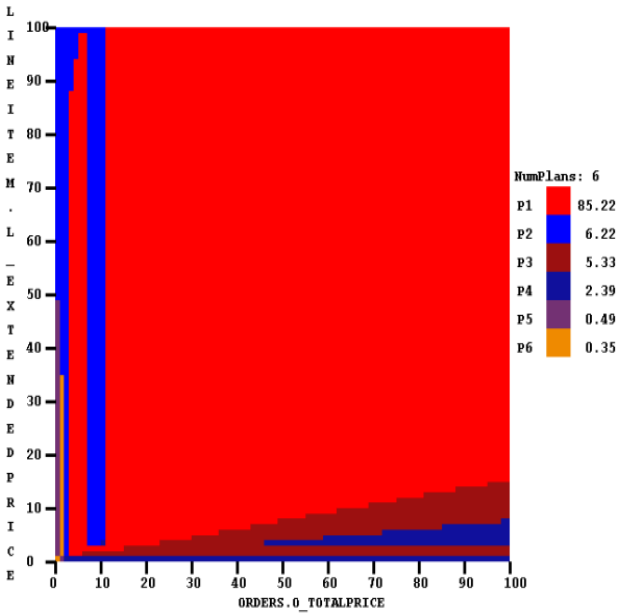
### 1.0.3 Organization

In Chapter 2 we will briefly discuss the related work.

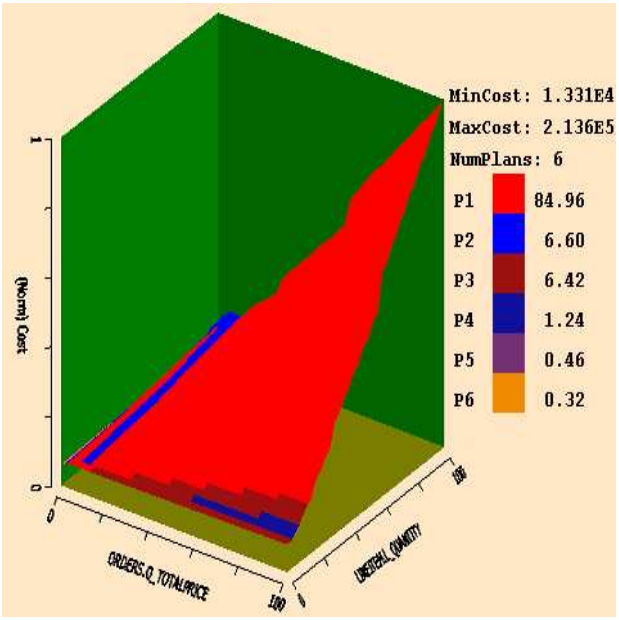
We will discuss the details of porting Picasso to new database system in Chapter 3. Section 3.1 will describe the testbed environment. In Section 3.2 we will present the porting of Picasso on Sybase. Porting on PostgreSQL is discussed in Section 3.3. And then in Section 3.4 we will briefly analyze OptD results and Section 3.5 will describe the results obtained on OptE.

Cardinality estimations of various databases have been analyzed in Chapter 4. Chapter 5 discusses the analysis of plan cardinality. In Section 5.1 we have given an analysis of variation of plan cardinality with respect to resolution. The analysis of plan cardinality reduction with increase in the cost threshold is explored in Section 5.2.

Finally in Chapter 6 we summarize our conclusions and discuss some future avenues.

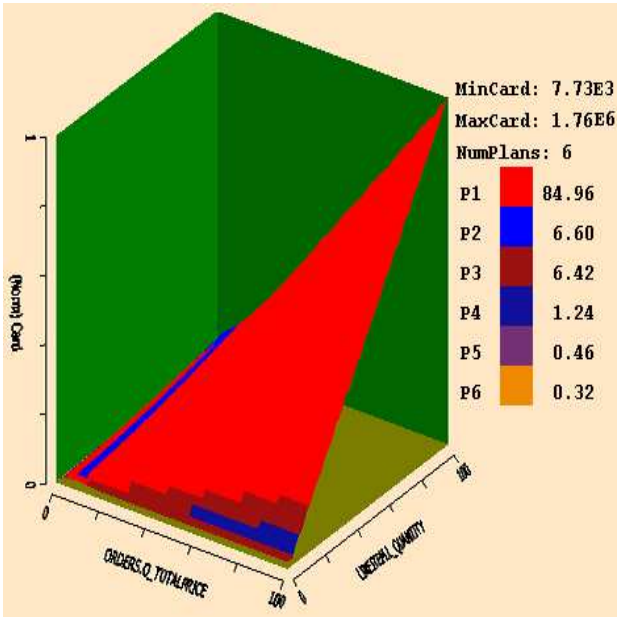



(a) **Plan Diagram**

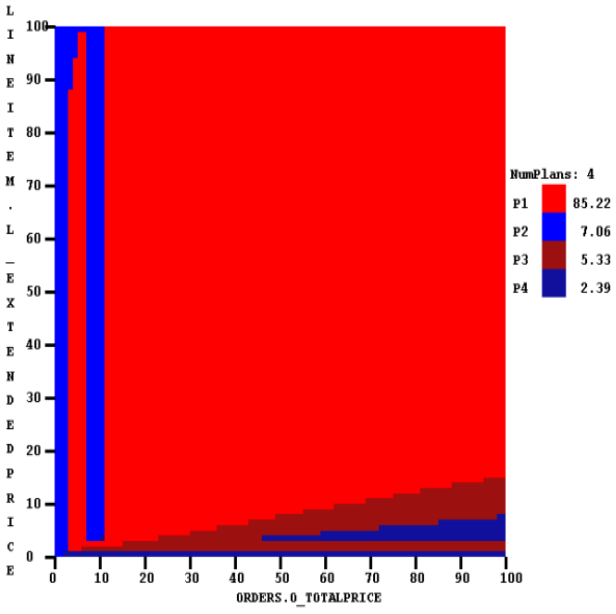



(b) **Plan-Cost Diagram**

Figure 1.3: Query 3P OptB




(a) **Plan-Card Diagram**




(b) **Reduced Plan Diagram**

Figure 1.4: Query 3P OptB

# Chapter 2

# Related Work

Various plan, plan-cost and plan-cardinality diagrams of three industrial strength query optimizers have been analyzed in [7] on TPC-H based queries. The results can be seen on the Picasso webpage [11]. The summary of observations made in [7] is presented here:

**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 [13], which ranges over [0,1], to quantify the skew, it was found that all the optimizers exhibit a marked skew in excess of 0.5 for most queries, on occasion going even higher than 0.8.

Overall, this leads to the hypothesis that current optimizers may perhaps be over-sophisticated 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. These complex patterns appear to indicate the presence of strongly non-linear and discretized cost models, again

perhaps an over-kill.

**Non-Monotonic Cost Behavior:** There were 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. 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, 3, 4, 9]. 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 do not hold true, *even approximately*, in the plan diagrams produced by the commercial optimizers.

# Chapter 3

# Porting Picasso to New Databases

## 3.1 Testbed Environment

### 3.1.1 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. Out of these 22 queries we have taken 17 queries. Two of the queries were one dimensional hence not used and three others which we left were not convertible to Picasso compatible format. 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.

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.

### 3.1.2 Relational Engines

The relational engines already evaluated in [7] are IBM DB2 v8.1 [17], Oracle 9i [19] and Microsoft SQL Server 2000 [18]. The new relational engines that have been evaluated in this report are Sybase ASE 15.0 [22] and PostgreSQL 8.0 [20].

### 3.1.3 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 relational engines were all installed with their default configurations for all parameters.

## 3.2 Picasso on Sybase

Sybase, Inc. has recently introduced Sybase Adaptive Server Enterprise (ASE) 15 [22] to meet the increasing demands of large databases and high transaction volumes, while providing a cost effective database management system. ASE 15 features new query processing technology that drastically increases performance thresholds and reduces hardware resource consumption.

Sybase was the remaining commercial query optimizer on which Picasso was not available. Hence to complete the spectrum of industrial strength and widely used database available with Picasso we worked on Sybase ASE 15.0. In this section we will discuss the various challenges that we faced for porting Picasso on Sybase.

The main tasks involved for this successful porting were:

- Find the statistics that Sybase maintains about the database.
- Extract relevant portions of the plan output generated by Sybase. There are mainly two variants of output: plain text and XML.

### 3.2.1 Statistics Access

All the statistics about any relation is present, as expected, in relational form in Sybase [14]. That means the meta-data about a relation is itself represented in a relation. The commonly required statistics of a relation are its cardinality the distribution of data in a column. We also need attributes of a table and their types. But the way this data is stored in Sybase is very complicated. We will elaborate this problem later.

To get information about a relation the system tables used were *systypes*, *sysobjects*, *syscolumns* [14]. The *systabstats* and *sysstatistics* [14] tables store statistics for all tables, indexes, and any un-indexed columns for which the user has explicitly created statistics. In general terms:

- *systabstats* stores information about the table or index as an object, that is, the size, number of rows, and so forth. It is updated by query processing, data definition language, and **update statistics** commands.
- *sysstatistics* stores information about the values in a specific column. It is updated by data definition language and **update statistics** commands

The *optdiag* utility displays statistics from the *systabstats* and *sysstatistics* tables. *Optdiag* can also be used to update *sysstatistics* information. We can use *optdiag* to display statistics for an entire database, for a single table and its indexes and columns, or for a particular column.

Histograms are created when *update statistics* command is executed. *Optdiag* utility can be used to display summary data about histograms. Histogram output is printed in columns,

- Step, the number of the steps.
- Weight, weight of each step.

- Operator, $<$ , $<=$, or $=$, indicating the limit of the value. Operators differ, depending on whether the cell represents a range cell or a frequency cell.
- Value, upper boundary of the values represented by a range cell or the value represented by a frequency count.

The problem with *sysstatistics* table is that the column names are like c0..c79 of type *varbinary* which is not interpretable. There is no documentation which gives any idea about what there columns store. This problem was overcome by using a stored procedure called *hist_values* provided by Sybase forum [23] which simulates these histogram tables. It takes the relation name and the attribute name as parameters and produces a nice interface, called histogram table with the same columns as mentioned earlier in the *optdiag* output, for accessing the histogram values.

### 3.2.2 Extracting Plan Information

Sybase like every other database system provides the user the facility to view the plan the optimizer has chosen as the best plan for executing the given SQL query. Whereas some databases choose to represent this information in relational format, following everything is a relation concept, others like Sybase provide this information in a textual format. We have to execute the following command to get the plan information:

```
set option show_abstract_plan on
```

Also if you do not wish to execute the query you can use the command:

```
set noexec on
```

Sybase generates this information in textual format and not in tables. So we had to come up with a parser to get the relevant information about a plan. But we were able to partially save this trouble of writing the parser as Sybase provides a *XML* output of the plan information. Thus we could easily use already available *SAX*(Simple API for XML) [21] parser and get the relevant information.

Here are a set of commands one needs to execute to get the XML output. The output has been shown later.

```
dbcc traceon(3604)
use tpch
set plan for show_best_plan_xml
    to client on
set noexec on
go
select * from REGION go exit

                    XML Output
-----------------------------------------------------------


<Optimizer-Trace>
    <Best-Global-Plan>
        <Pop>
            <PopEmit>
                <PopIndScan>
                Gtt0(REGION)
                Gti0(REGION_6045)
                </PopIndScan>
            </PopEmit>
        </Pop>
    </Best-Global-Plan>
</Optimizer-Trace>
```

The XML output consists of various tags and the nesting of elements is used to represent hierarchy. The various operations in Sybase XML output are preceded by the word *Pop*. The plan output here displays that an index scan of table REGION using index REGION_6045 has been done to retrieve data.

As we mentioned earlier that we were able to save the trouble of writing parser partially. It is due to the fact that the XML output provides only the plan tree information but in order to get the cost and cardinality of each operator we had to use the textual output displayed in the long format. The following command is used to generate textual output with full details:

```
set option show_best_plan long
```

The relevant portion of the output for this command is shown here,

```
FINAL PLAN ( total cost = 54.5 ):
    lio=2 pio=2 cpu=5
( PopEmit
    proj:  REGION.R_COMMENT  , REGION.R_NAME  , REGION.R_REGIONKEY
    .
    .
    cost: 0
    I/O estimate : [
        rowcount=5
    .
    .
    ( PopIndScan index: Gti0( REGION_6080021661 )
            table: Gtt0( REGION )
        .
        .
        .
        cost: 54.5
        I/O estimate : [
                        rowcount=5
```

```
explain select
    l_shipmode
from
    orders, lineitem
where
    o_orderkey <= l_orderkey
    and o_totalprice <= 158085
    and l_quantity < 39;


                            QUERY PLAN
------------------------------------------------------------------


 Hash Join
 (cost=55594.39..349199.94
        rows=283799 width=14)
   Hash Cond: ("outer".l_orderkey =
            "inner".o_orderkey)
   ->  Seq Scan on lineitem
       (cost=0.00..261789.26
                rows=510824 width=18)
         Filter: ((l_quantity <
                    39::numeric))
   ->  Hash
       (cost=50255.00..50255.00
                rows=833356 width=4)
         ->  Seq Scan on orders
             (cost=0.00..50255.00
                rows=833356 width=4)
             Filter:(o_totalprice <
                158085::numeric)
(7 rows)
```

Figure 3.1: Explain Command in PostgreSQL

## 3.3 Picasso on PostgreSQL

PostgreSQL is an industrial strength Object Relational Database Management System (ORDBMS). It supports a large part of the SQL-2003 standard and offers many modern features

like complex queries, foreign keys, triggers, views, transactional integrity, multi-version concurrency control, etc. Picasso was ported on PostgreSQL and an analysis was done of its query optimizer through various plan, cost and cardinality diagrams as shown in [5]. This is the first open-source database which has been analyzed using Picasso. The main tasks were similar to those which we enlisted in case of Sybase:

- Access the statistics that PostgreSQL maintains about the database.
- Extract relevant portions of the textual plan output.

### 3.3.1 Statistics Access

The most important catalog tables that were required are *pg_class*, *pg_am*, *pg_statistic*, *pg_type*, *pg_stats* , *pg_tables* and *pg_indexes*. Another important catalog entry is that of the histograms. PostgreSQL maintains a histogram catalog which is a mixture of end-biased [10] and equi-depth [10] histograms. Here, the histograms are stored in the *pg_stats* catalog table, which stores both the most frequently used values as well as rest of the values.

### 3.3.2 Extracting Plan Information

Like any other database the plan chosen by the PostgreSQL optimizer is available for the user viewing. But like Sybase the output is textual. In order to get the plan information PostgreSQL has provided *explain* command. Example is as shown in Figure3.1.

To get the relevant information from the plan a parser was written. The parser is very naive and has been developed using Java *String* class functions [15].

## 3.4 Result Analysis of OptD

An analysis of various interesting diagrams on OptD has been presented in this section:

### 3.4.1 Plan Cardinality Analysis

Average plan cardinality in case of OptD is 14.4%, see Table 3.1, over dense queries which is lower than other optimizers considered for analysis.

### 3.4.2 Duplicates and Islands

There were instances of plan diagrams where duplicates and islands were observed in case of OptD like other optimizers. Total number of duplicates and islands present in plan diagrams for OptD has been shown in Table 3.3. It has least number of duplicates, 45, amongst all optimizers which further reduces to 10 after plan diagram reduction.

### 3.4.3 Plan Cardinality Reduction

Plan cardinality reduction experiments as in [7] were performed for OptD and results are shown in Table 3.2. The cost increase threshold $\lambda$ was set to 10%. Very significant reductions were obtained on OptD with average over dense queries being 80%. And like other optimizers the average cost increase, shown in second column of Table 3.2, is much less than the threshold value of 10%. Although the plan diagrams of OptD were simple as compared to other optimizers still the amount of reduction achieved suggests that there is still lot of room for improvement in the optimizer design.

## 3.5 Result Analysis of OptE

After the porting of Picasso on OptE was accomplished the most important work was to analyze the diagrams generated by Picasso. We will discuss various observations made in this section.

### 3.5.1 Plan Space Coverage Skew

We start our analysis by considering the skew in the space coverage by optimal plan set chosen by the optimizer over the entire selectivity space. Table 3.1 displays the results for OptE along with other optimizers on queries based on TPC-H which were found dense in any of the

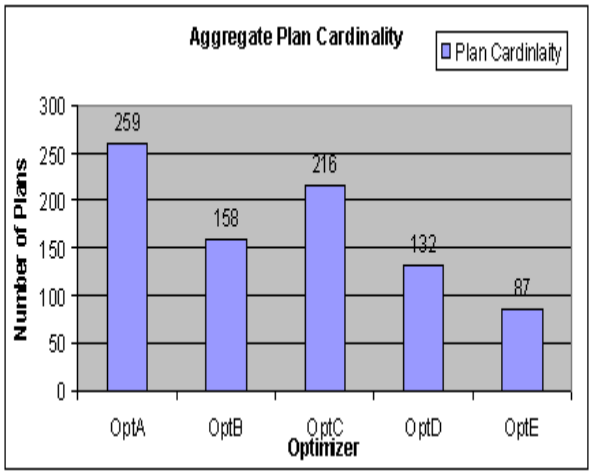



Figure 3.2: Aggregate Plan Cardinality over all TPC-H queries

optimizers. The non-dense queries have been shown in gray color. The data for OptA, OptB and OptC is from [7]. The first column that shows cardinality of the optimal plan set reveals something quite astonishing. As evident from the table in case of OptE *only one query*, Query 10P, qualifies as dense query (having 10 or more plans in the plan diagram). The plan diagram for Query 10P is shown in Figure 3.3. That too it is a borderline case as it has only 11 plans. This has made OptE to fall in an altogether different league of commercial optimizer as till date whichever commercial optimizer we had investigated each one had the plan cardinality average more than 20. Figure 3.5 shows the aggregate plan cardinality for all the databases over all queries (dense as well as non-dense). Here it is surprising that OptE is about one-third of OptA. Consider the case of Query 9P for which all commercial optimizers have plan cardinality in excess of 40 but for OptE this number is only 4!

One reason for low plan cardinality of OptE may be due to the fact that we are considering plans at a very coarse level. There are certain sub-operator level details which are not provided by the output. Thus it might be the case that when we consider two plans as same they might be differing at the lower level of information. These details are present in the long textual output but the problem is we do not have a parser to obtain the information from that output.

When the fraction of optimal plan set required to cover 80% area is considered for OptE, we see that the average (over all queries) is around 35% as shown in second column of Table 3.1.

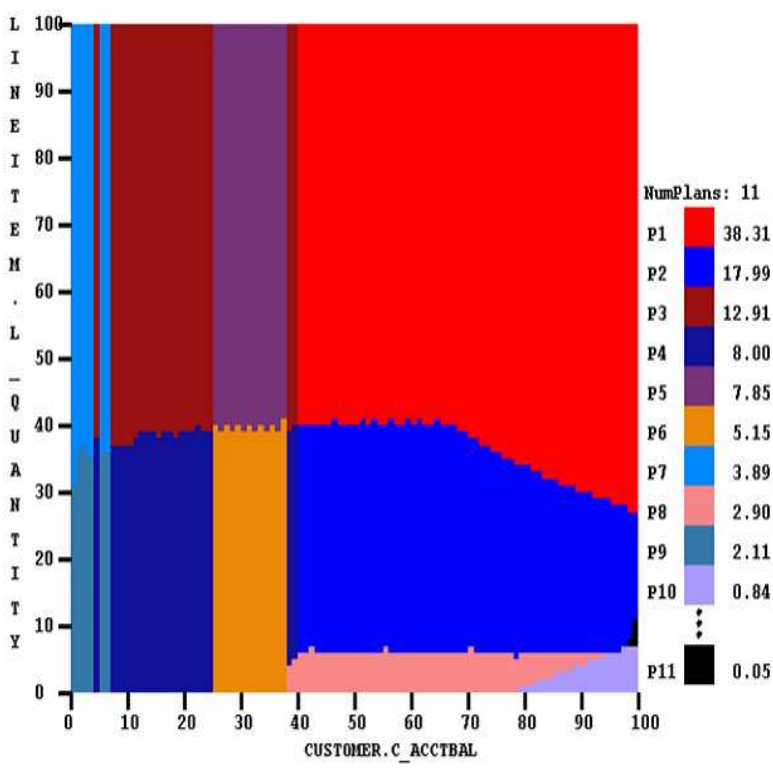



Figure 3.3: Plan Diagram (Query 10P OptE)

Third column of Table 3.1 is for Gini index [13]. Gini index is bounded on the interval [0, 1], with 0 representing no skew and 1 representing extreme skew. Here OptE is consistent with other database optimizers and has Gini index values in excess of 0.5 which represents high skew.

### 3.5.2 Plan Cardinality Reduction

While introducing the Picasso tool we discussed that an algorithm for plan cardinality reduction was presented in [7]. With similar objective in mind we proceeded with OptE experiments for plan cardinality reduction. The results are shown in Table 3.2. The cost increase threshold $\lambda$ was set to 10%. Very significant reductions were obtained on OptE, like other optimizers, with average over all queries being 53%. And like other optimizers the average cost increase, shown in second column of Table 3.2, is 1.2% which is much less than the threshold value of 10%.

| TPC-H Query Number | OptA | | | OptB | | | OptC | | | OptD | | | OptE | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Plan Card | 80% Coverage | Gini Index | Plan Card | 80% Coverage | Gini Index | 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 | 9 | 45% | 0.79 | 4 | 25% | 0.58 |
| 3 | 15 | 26% | 0.72 | 6 | 17% | 0.74 | 6 | 50% | 0.50 | 10 | 10% | 0.84 | 4 | 50% | 0.58 |
| 5 | 21 | 19% | 0.81 | 14 | 21% | 0.74 | 18 | 17% | 0.81 | 16 | 12% | 0.81 | 5 | 20% | 0.76 |
| 7 | 13 | 23% | 0.73 | 6 | 50% | 0.46 | 19 | 15% | 0.79 | 15 | 33% | 0.67 | 9 | 11% | 0.86 |
| 8 | 31 | 16% | 0.81 | 25 | 25% | 0.72 | 38 | 18% | 0.79 | 25 | 26% | 0.81 | 7 | 43% | 0.56 |
| 9 | 63 | 9% | 0.88 | 44 | 27% | 0.70 | 41 | 12% | 0.83 | 10 | 33% | 0.74 | 4 | 50% | 0.55 |
| 10 | 24 | 16% | 0.78 | 9 | 22% | 0.69 | 8 | 25% | 0.75 | 11 | 18% | 0.75 | 11 | 36% | 0.55 |
| 16 | 12 | 42% | 0.58 | 3 | 67% | 0.25 | 4 | 25% | 0.72 | 6 | 33% | 0.59 | 7 | 29% | 0.67 |
| 18 | 5 | 60% | 0.33 | 13 | 38% | 0.57 | 5 | 20% | 0.75 | 14 | 29% | 0.69 | 8 | 25% | 0.79 |
| 21 | 27 | 22% | 0.74 | 6 | 17% | 0.80 | 22 | 18% | 0.81 | 5 | 20% | 0.79 | 3 | 66% | 0.21 |
| **Avg(dense)** | **25.3** | **21%** | **0.76** | **22.0** | **23%** | **0.72** | **28.8** | **16%** | **0.8** | **14.4** | **21.6%** | **0.76** | **11** | **36%** | **0.55** |

Table 3.1: Skew in Plan Space Coverage

| TPC-H Query Number | OptA | | | OptB | | | OptC | | | OptD | | | OptE | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Percent Card Decr | Avg Cost Incr | Max Cost Incr | Percent Card Decr | Avg Cost Incr | Max Cost Incr | Percent Card Decr | Avg Cost Incr | Max Cost Incr | Percent Card Decr | Avg Cost Incr | Max Cost Incr | Percent Card Decr | Avg Cost Incr | Max Cost Incr |
| 2 | 59.2 | 1.0 | 4.4 | 64.2 | 0.6 | 5.9 | 77.1 | 3.2 | 6.4 | 25 | 3.2 | 6.4 | 0 | 0 | 0 |
| 3 | 73.3 | 1.9 | 8.1 | 33.3 | 3.5 | 7.0 | 33.3 | 1.1 | 1.24 | 64 | 1.1 | 1.24 | 25 | 0.8 | 0.8 |
| 5 | 67.3 | 2.6 | 8.1 | 42.9 | 0.1 | 0.6 | 61.1 | 0.2 | 8.1 | 46 | 0.2 | 8.1 | 80 | 0.64 | 5.6 |
| 7 | 46.1 | 0.1 | 9.5 | 16.6 | 0.4 | 0.7 | 54.5 | 1.1 | 9.5 | 57 | 1.1 | 9.5 | 88.8 | 1.5 | 8.6 |
| 8 | 87.6 | 0.4 | 9.4 | 84.0 | 0.9 | 9.1 | 86.8 | 1.2 | 8.4 | 84 | 1.2 | 8.4 | 85.7 | 1.4 | 7.5 |
| 9 | 84.4 | 1.6 | 8.6 | 36.4 | 1.4 | 8.9 | 80.5 | 2.1 | 8.3 | 75 | 2.1 | 8.3 | 75 | 0.1 | 0.1 |
| 10 | 67.6 | 0.8 | 4.4 | 44.4 | 0.5 | 6.1 | 62.5 | 0.4 | 2.4 | 50 | 0.4 | 2.4 | 63 | 2.2 | 8 |
| 16 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 20 | 0.8 | 1.2 | 28.5 | 0.7 | 2.9 |
| 18 | 40.0 | 0.1 | 0.5 | 46.2 | 3.7 | 9.6 | 0.0 | 0.0 | 0.0 | 41 | 2.5 | 3.7 | 50 | 2.3 | 6.3 |
| 21 | 59.8 | 0.0 | 0.2 | 66.7 | 0.9 | 2.5 | 68.2 | 0.7 | 6.9 | 0.0 | 0.0 | 0.0 | 33.3 | 3.25 | 6.1 |
| **Avg(dense)** | **60.6** | **0.9** | **5.6** | **56.9** | **0.7** | **6.1** | **71.4** | **1.4** | **7.9** | **59.6** | **1.4** | **7.9** | **63** | **2.2** | **8** |

Table 3.2: Plan Cardinality Reduction

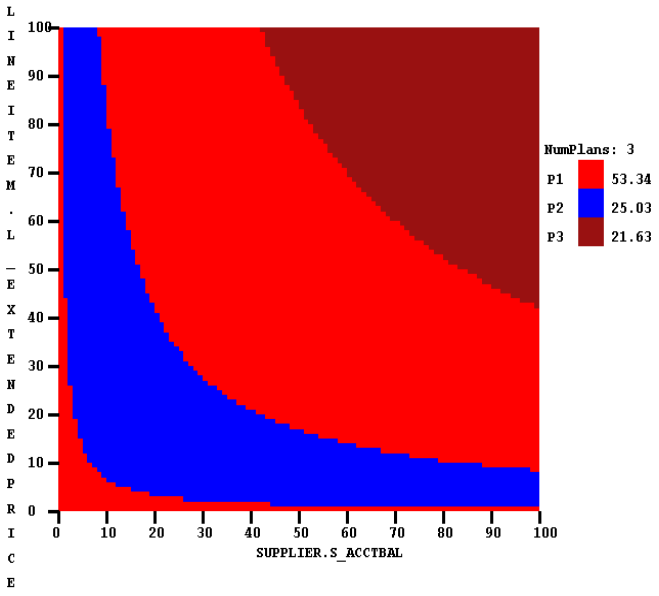



Figure 3.4: Plan Diagram (Query 21P OptE)

### 3.5.3 Relationship to PQO

The assumptions which form the basis of Parametric Query Optimization do not hold true in practice for various commercial optimizers as discussed in Chapter 2. Similar cases of violation have been observed in OptE as well.

- Plan Convexity is violated by plan P1 and P2 in Figure 3.4.
- Plan Uniqueness is violated in Figure 3.4 by plan P1 which appears twice.
- Plan Homogeneity is violated by plan P5 which forms islands within plan P1 as illustrated in Figure 3.5.

### 3.5.4 Interesting Pictures Gallery

We came across some interesting diagrams during this analysis which are presented in this section.

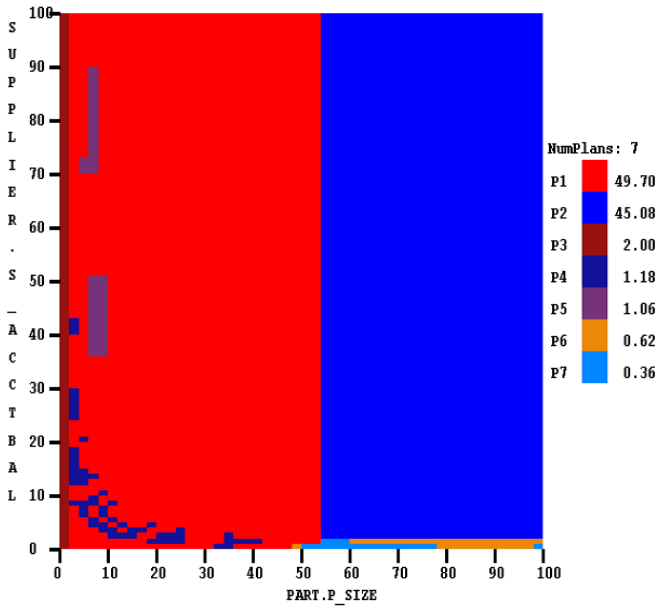



Figure 3.5: Plan Diagram (Query 16P OptE)

| Databases | # Duplicates | | # Islands | |
|---|---|---|---|---|
| | Original | Reduced | Original | Reduced |
| OptA | 136 | 14 | 38 | 3 |
| OptB | 80 | 15 | 1 | 0 |
| OptC | 55 | 7 | 8 | 3 |
| OptD | 45 | 8 | 10 | 2 |
| OptE | 151 | 43 | 131 | 35 |

Table 3.3: Duplicates and Islands

### Duplicates and Islands

Table 3.3 gives the total number of duplicates and islands present in plan diagrams for each optimizer over all the queries of TPC-H benchmark. OptE has more number of duplicates and islands than any other database. After plan reduction has been applied we see a drastic amount of reduction in duplicates and islands almost one-third in case of OptE. By duplicate we mean that the same plan may appear at a different disjoint location. Plan P1 is present at two places in Figure 3.4. Also we can see Plan P5 present as an island in the sea of Plan P1 in Figure 3.5.

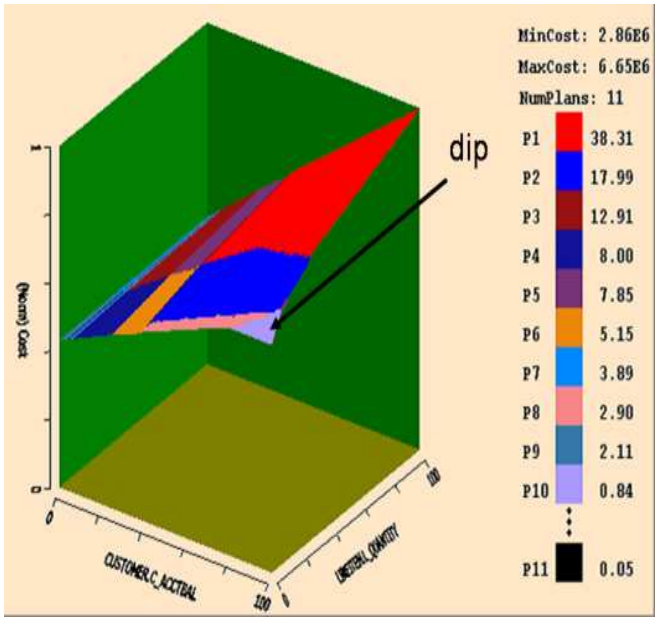



Figure 3.6: Plan-Cost Diagram (Query 10P OptE)

### Plan Switch Point

In many plan diagrams of all the optimizers we saw a line running parallel to selectivity axis through the entire selectivity space with a plan *switch* occurring for all plans bordering the line, as we move across the line. In Figure 3.3 we have a line at 25% selectivity of CUSTOMER table. The change across the line is in the join order and join operator. On left side of the line we have NATION $\triangleright_{hashjoin}\triangleleft$ CUSTOMER $\triangleright_{hashjoin}\triangleleft$ ORDERS which is reordered to CUSTOMER $\triangleright_{mergejoin}\triangleleft$ ORDERS $\triangleright_{hashjoin}\triangleleft$ NATION on right side of the line.

### Non-Monotonic Cost Behavior

We had observed non-monotonic cost behavior in all optimizers. OptE also exhibits this behavior but not to a large extent. In Figure 3.6 we can see that Plan P10 is having a dip.

### Fence Pattern

For Query 5P we got Figure 3.7 and it had an unusual pattern which resembles the *fence*. We can see Plan P2 interleaved within Plan P1 as vertical bars of equal size. The difference between

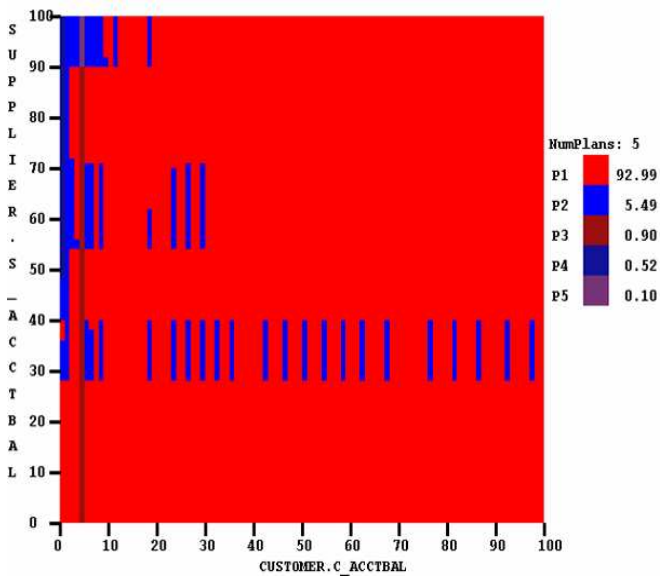



Figure 3.7: Plan Diagram (Query 5P OptE)

the plans was that SUPPLIER table was outer relation in a join in one of the plans but switched to inner relation in the other. Also worth mentioning is the fact that the costs of Plan P1 and P2 do not differ much! After reduction we got a single plan.

### Confetti Pattern

Figure 3.8 was obtained by Picasso with Query 11P on OptE. We can see a curve above which there is Plan P2 sprinkled all over P1 and Plan P6 sprinkled on P5. The difference between the plans is in the order of *sequence* and *sort* operators. But again the cost is almost flat across whole selectivity space!

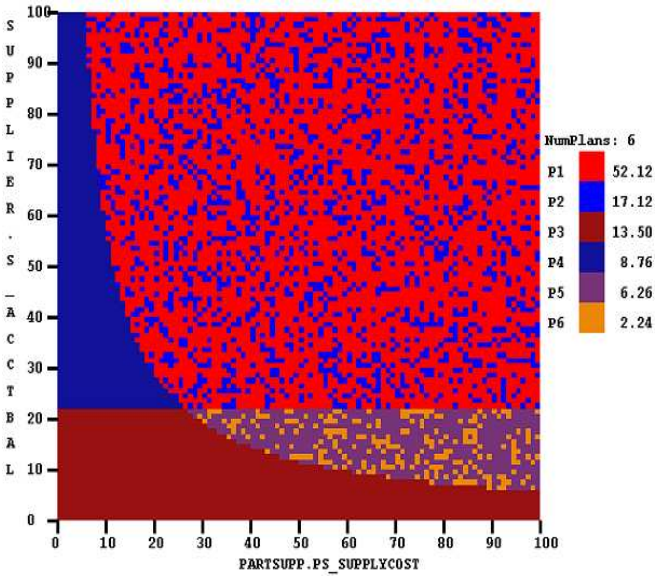



Figure 3.8: Plan Diagram (Query 11P OptE)

# Chapter 4

# Cardinality Estimate Analysis

We mentioned earlier that the query optimizers try to estimate the result cardinality of the given query. The plan-cardinality diagrams generated for same query over different query optimizers were quite different and hence we did some analysis on the cardinality estimates of query optimizers. We have analyzed inter-optimizer and intra-optimizer estimates. Inter-optimizer comparison was possible as the unit of estimation for cardinality is same, number of tuples, across all optimizers unlike cost. For intra-optimizer we have used OptA and worked on different optimization levels.

We will first discuss the inter-optimizer differences. Figure 4.1 summarizes our results. We have considered queries which had high estimated cardinality value. So the queries which have simple aggregate function like *count* and have lots of *group by* which produce low cardinality output have not been included in the analysis. The queries have been numbered like 2min, 2max, 3min and 3max etc. Query 2min represents Query 2P from TPC-H which results in minimum cardinality estimate over whole selectivity space. Similarly Query 2max is for maximum cardinality estimate. The x-axis in Figure 4.1 is for queries and for each query we have 5 bars representing OptA, OptB, OptC, OptD and OptE. The y-axis represents the estimation done by optimizer as a percentage of actual result cardinality on a log scale.

One can easily observe that there is no consensus amongst the optimizers on the estimated result cardinality. The difference in their estimates is huge and also off by a considerable margin from the actual cardinality! Only in one case, Query 14max, all the optimizers estimated the

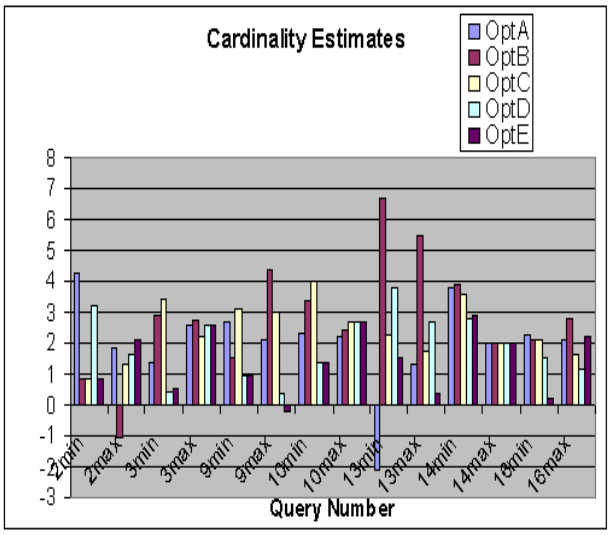
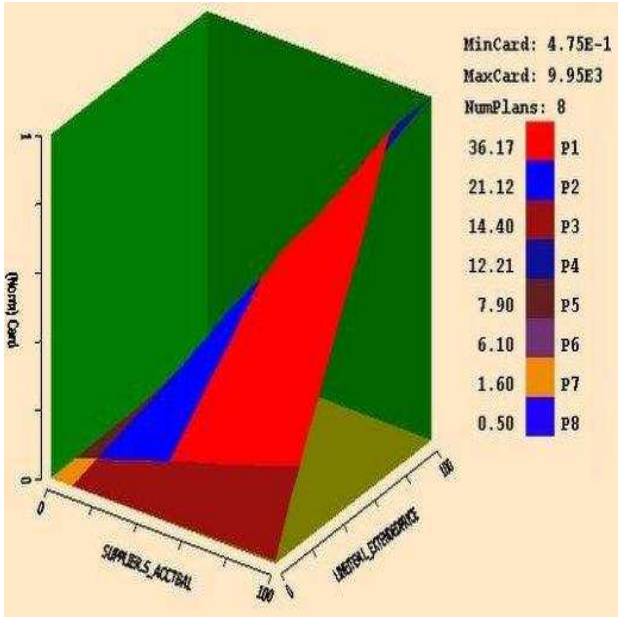


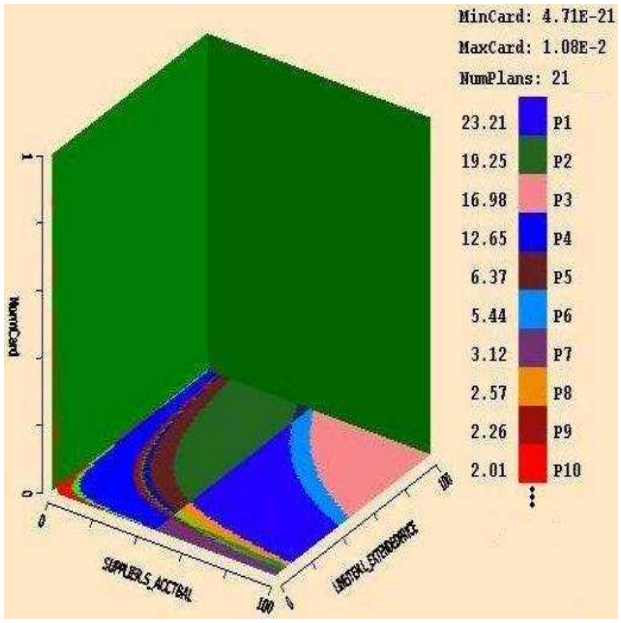

Figure 4.1: Cardinality Estimation Errors

cardinality very accurately as shown by the log scale value 2 which represents 100%. Also all the optimizers do both under-estimation and over-estimation of the result cardinality.

Another surprising result was obtained when we did intra-optimizer analysis for cardinality estimate. As we mentioned earlier we have used 3 different levels of optimization for OptA. It seemed very unlikely that the cardinality estimation module will behave differently in different optimization levels. But we observed very huge differences in the estimates of OptA at lowest and default. But default level and highest level estimates matched which was the expected behavior. We have shown in Figure 4.2(a) and 4.2(b) the plan-cardinality diagrams for Query 21P on OptA lowest level and default level respectively. The diagram for highest level is same as default level and hence is not shown here. We can easily see that there is quite enormous difference in the OptA's estimate for different optimization levels. A very startling fact is that lowest optimization level estimates are much more accurate for this query than other levels considered.




(a) **Lowest level**




(b) **Default level**

Figure 4.2: Plan-Cardinality Diagram (Query 21P OptA)

# Chapter 5

# Results on Plan Cardinality

## 5.1 Plan Cardinality Distribution

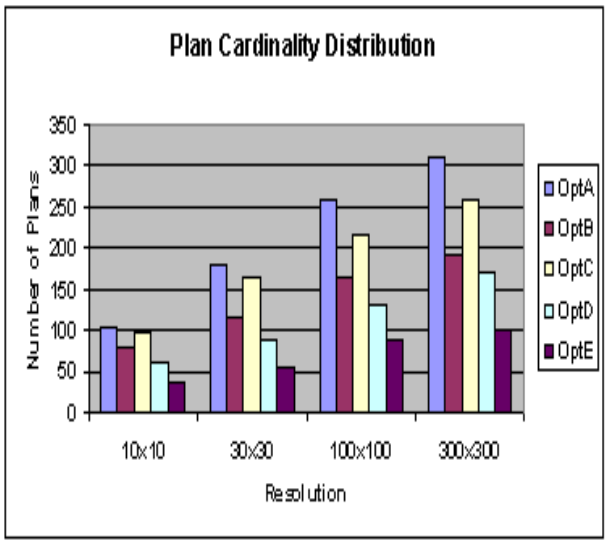



Figure 5.1: Plan Cardinality Distribution (Over all queries)

In Picasso [7] the diagrams were generated by varying the selectivity on two different relations. The diagrams presented in that paper were obtained by varying selectivity from [0.5%, 0.5%] to [99.5%, 99.5%]. To achieve this 100 queries were fired along each dimension, making

it 10,000 queries in all. As mentioned in [7] there was a significant increase in number of plans if finer resolution was used. This observation led us to a series of experiment across 5 different industrial strength optimizers.

The general trend observed was that all optimizers exhibited an increase in the plan cardinality with the increase in resolution as shown in Figure 5.1. The x-axis represents the query grid resolution and y-axis the plan cardinality. For this experiment the query grid granularities used were 10x10, 30x30, 100x100, and 300x300 where, in $N$ x $N$, $N$ represents the number of queries fired in each dimension. The number of plans displayed in the graph is cumulative sum over all queries based on TPC-H.

It is interesting to see the behavior of OptA. As it is evident from the graph the skew in plan cardinality is maximum in OptA and it seems impossible to claim that after a certain resolution there will not be any increase in the plan cardinality. Although the plan cardinality in case of OptE is low still if percentage wise increase in plan cardinality is considered then OptE is comparable to other databases. Thus it can be concluded that optimizers are really making very fine grained choices.

## 5.2 Plan Cardinality Reduction Statistics

We have been discussing about plan diagram reduction technique throughout this report. It is one of the important aspects of Picasso as it provides some direction which may lead to simplification of optimizer design.

We had discussed about cost increase threshold denoted by $\lambda$. It was set to 10% for all the previous experiments. Now a question to ask is, *"Is there a value of $\lambda$ which gives maximum achievable reduction?"*. To answer this intriguing question we plotted the number of plans against the cost increase threshold. Figure 5.2 displays the graph. The x-axis is the cost increase threshold and y-axis is the number of plans. The experiment was done on all five optimizers we have discussed so far.

The value of $\lambda = 0$ represents original plan diagram without any reduction. We see that there is sharp fall in number of plans as we increase the threshold. Most of the plan diagrams

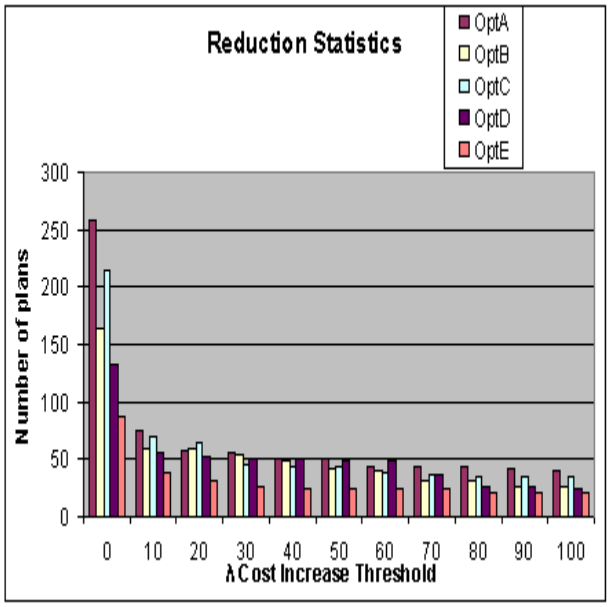



Figure 5.2: Reduction Statistics (Over all queries)

had single plan after reduction with 70% threshold. As visible from Figure 5.2 the knee of the graph for all databases is around 10%. This highlights the fact that even if we increase threshold beyond 10% there is not much reduction benefit obtained than what is achieved with 10% threshold.

# Chapter 6

# Conclusions and Future Work

The unnecessary complexity prevailing in the current commercial query optimizers was pointed out by various Picasso diagrams. Sybase and PostgreSQL the two remaining widely used databases were made available on Picasso, which has completed the spectrum of industrial strength query optimizers.

Heavy skew in plan space coverage area was observed in both OptD and OptE. OptE had some peculiar diagrams like *fence* and *confetti* pattern. Although not much variation in cost diagram behavior has been observed in case of OptE. OptE had surprisingly very low plan cardinalities!

Cardinality estimates made by all optimizers were found to be off by huge margin. In case of OptA the estimates varied with optimization levels. Also we saw the distribution of plan cardinality as we vary the resolution and it seems that optimizers are trying to make very fine-grained choices. Also 10% cost increase threshold seems to be reasonable enough as shown in the discussion since beyond that point not much reduction is achieved.

In future we would like to see the results on 1000x1000 resolution as it will be finest possible resolution practically. Since Sybase allows user to feed the plan which the user wants, in form of *abstract plans*, we can try feeding plans of other databases into Sybase and cost them for comparison. Also if TPC-DS [6], a new decision support system benchmark, is released then it will be of great interest to run Picasso on that dataset as it has very complex queries.

# Bibliography


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

[2] 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.

[3] 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.

[4] 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.

[5] P. Mandal, "PostgreSQL on Picasso : Implementation and Analysis", *Mid-term project report, Dept. of Computer Science and Automation, IISc Bangalore*, December 2005

[6] M. Poess, B. Smith, L. Kollar, and P. Larson, "TPC-DS, Taking Decision Support Benchmarking to the Next Level", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2002

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

[8] N. Reddy, "Next Generation Relational Query Optimizers", *Master's Thesis, Dept. of Computer Science and Automation, IISc Bangalore*, June 2005.

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

[10] Y. Ioannidis, V. Poosala. *"Histogram-Based Solutions to Diverse Database Estimation Problems"*. Data Engineering Bulletin, 1995.

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

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

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

[14] *http://infocenter.sybase.com/*

[15] *http://java.sun.com/j2se/1.4.2/docs/api/index.html*

[16] *http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp*

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

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

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

[20] *http://www.postgresql.org*

[21] *http://www.saxproject.org/*

[22] *http://www.sybase.com/products/informationmanagement/adaptiveserverenterprise*

[23] *http://www.sybase.com/support/newsgroups*

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