Reduction of Query Optimizer Plan Diagrams

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

By

Pooja N. Darera

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

December 2007

Acknowledgements

First and foremost, I would like to thank my guide Prof. Jayant Haritsa for his enduring support, encouragement and motivation. I would also like to thank my parents and brother for their support and guidance.

A special thanks to Harish, who collaborated with me on this work. I would also like to thank the members of DSL, SSL, Rythmica and Papyrus for making IISc a really fun place to be in. I'm also grateful to my friends Soumya, Shreya, John, Vipul and Paneendra for always being extremely supportive. A huge thanks to Dr. Vathsala for her unwavering patience and encouragement through difficult times.

Last and by no means the least, I would like to thank Suresh, without whose invaluable support none of this would have been possible.

Abstract

Modern database systems use a query optimizer to identify the most efficient strategy, called "plan", to execute declarative SQL queries. Optimization is a mandatory exercise since the difference between the cost of the best plan and a random choice could be in orders of magnitude. The role of query optimizers is especially critical for the decision-support queries featured in data warehousing and data mining applications.

For a query on a given database and system configuration, the optimizer's plan choice is primarily a function of the selectivities of the base relations participating in the query. A pictorial enumeration of the execution plan choices of a database query optimizer over this relational selectivity space is called a "plan diagram". It has been shown recently that these diagrams are often remarkably complex and dense, with a large number of plans covering the space. An interesting research problem that immediately arises is whether complex plan diagrams can be reduced to a significantly smaller number of plans, without materially compromising the query processing quality. The motivation is that reduced plan diagrams provide several benefits, including quantifying the redundancy in the plan search space, enhancing the applicability of parametric query optimization, identifying error-resistant and least-expected-cost plans, and minimizing the overhead of multi-plan approaches.

In this thesis, we investigate the plan diagram reduction issue from theoretical, statistical and empirical perspectives. Our analysis shows that optimal plan diagram reduction, w.r.t. minimizing the number of plans in the reduced diagram, is an NP-hard problem in general, and remains so even for a storage-constrained variation. We then present CostGreedy, a greedy reduction algorithm that has tight and optimal performance guarantees, and whose complexity scales linearly with the number of plans in the diagram. Next, we construct an extremely fast estimator, AmmEst, for identifying the location of the best tradeoff between the reduction in plan cardinality and the impact on query processing quality. Both CostGreedy and AmmEst have been incorporated in the publicly-available Picasso optimizer visualization tool.

Through extensive experimentation with a suite of representative multi-dimensional SQL query templates based on the TPC-H and TPC-DS benchmarks, executed on industrial-strength database optimizers, over a range of data distributions, query distributions, and memory budgets, we demonstrate the following: Complex plan diagrams can be very substantially reduced - in fact, they become "*anorexic*" i.e. reduced to a very small absolute number of plans - with only a marginal increase in query processing costs. While these results are produced using a highly conservative upper-bounding of plan costs based on a cost monotonicity constraint, when the costing is done on "actuals" using abstract plan features, the reduction obtained is even greater - in fact, often resulting in only a couple of plans in the reduced diagram. We also experimentally demonstrate some of the benefits of this reduction w.r.t. enhanced resistance to errors in selectivity estimates of optimizers and present an algorithm to produce reduced plan diagrams that are inherently resistant to selectivity estimate errors.

In summary, this thesis demonstrates that complex plan diagrams can be efficiently converted to highly anorexic reduced diagrams, a result that could have useful implications for the design and use of next-generation database query optimizers.

Publications

1. Harish D., Pooja Darera, Jayant Haritsa,
   "*On The Production of Anorexic Plan Diagrams*",
   Proc. of 33rd Intl. Conf. on Very Large Data Bases (VLDB), Vienna, Austria, September 2007.

2. Harish D., Pooja Darera, Jayant Haritsa,
   "*Reduction of Query Optimizer Plan Diagrams*", Technical Report, TR-2007-01, DSL/SERC, Indian Institute of Science
   http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2007-01.pdf

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Structured Query Language (SQL) [29] is the international standard for querying relational database management systems (DBMS) such as IBM's DB2, Microsoft's SQL Server, Oracle, etc., which form the cornerstone of today's information industry. SQL is a declarative language in the sense that an SQL query specifies *what* has to be done, not *how* it is to be done. A sample SQL query on the TPC-H benchmark schema [45] is given in Figure 1.1, which lists the mode of shipping for all items whose quantity is less than or equal to 20, which are a part of an order of price 100 or less.

```
select l_shipmode
from orders, lineitem
where o_orderkey = l_orderkey
      and o_totalprice ≤ 100
      and l_quantity ≤ 20
group by l_shipmode
order by l_shipmode
```

Figure 1.1: Sample SQL Query

Modern database systems use a *query optimizer* to identify the most efficient strategy to execute declarative SQL queries. The efficiency of the strategies, called "query execution plans" or simply "plans", is usually evaluated or costed in terms of the estimated query response time.

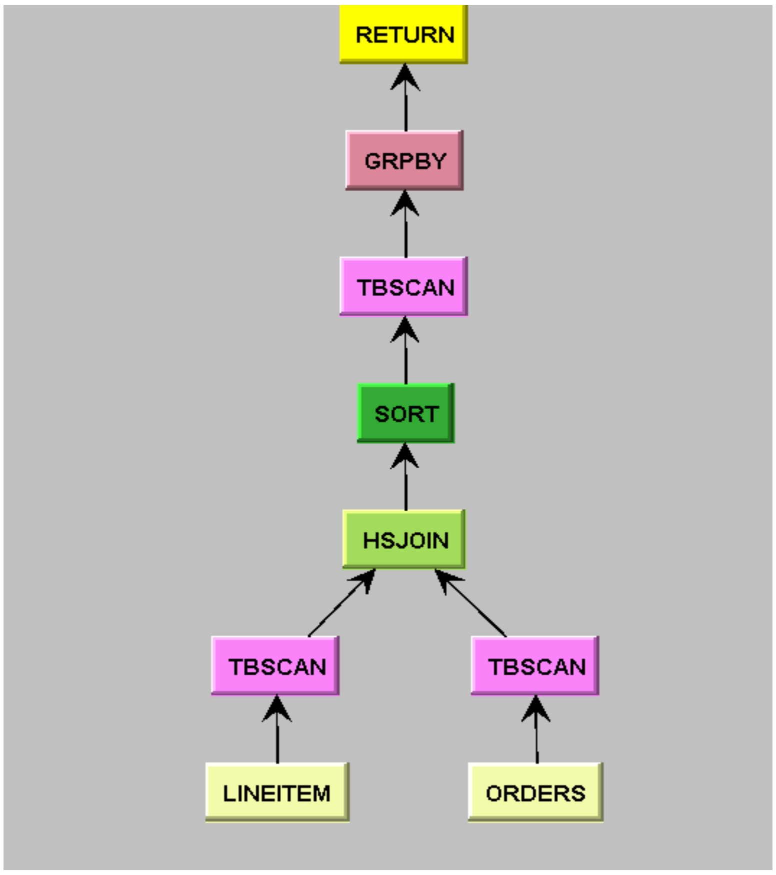


Figure 1.2: Query Execution Plan

A sample plan for the query given in Figure 1.1 is shown in Figure 1.2. This plan performs a sequential scan of the ORDERS and LINEITEM relations before joining them using the *hash join* operator. It finally sorts and groups the results in the required order.

Optimization is a mandatory exercise since the difference between the cost of the best plan and a random choice could be in orders of magnitude [44]. The role of query optimizers has become especially critical in recent times due to the high degree of query complexity characterizing current decision-support applications, as exemplified by the TPC-H benchmark [45], and its new incarnation, TPC-DS [46].

Query optimization is a difficult problem due to the large number of possible ways to execute a given query using different access methods, join orders, join operators, etc. While industrial strength query optimizers each have their own proprietary methods to identify the best plan, the de-facto standard underlying strategy is based on the classical System-R optimizer [38] proposed about three decades ago. This method is: Given a user query, first apply a variety of heuristics to restrict the combinatorially large search space of plan alternatives to a manageable size; then 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.

Query optimization using this cost-based approach is computationally expensive w.r.t. the time and resources that need to be expended to find the best plan. Therefore, understanding and characterizing query optimizers with the ultimate objective of improving their performance is a fundamentally important issue in the database research literature.

1.1 Query Templates

select o_year, sum(case when nation = ’BRAZIL’ then volume else 0 end) / sum(volume)

from (select YEAR(o_orderdate) as o_year, l_extendedprice * (1 - l_discount) as volume, n2.n_name as nation

from part, supplier, lineitem, orders, customer, nation n1, nation n2, region

where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey and o_custkey = c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey and s_nationkey = n2.n_nationkey and r_name = ’AMERICA’ and p_type = ’ECONOMY ANODIZED STEEL’
and **s_acctbal** $\leq C_1$ and **l_extendedprice** $\leq C_2$

) as all_nations

group by o_year

order by o_year

Figure 1.3: Example Query Template: QT8

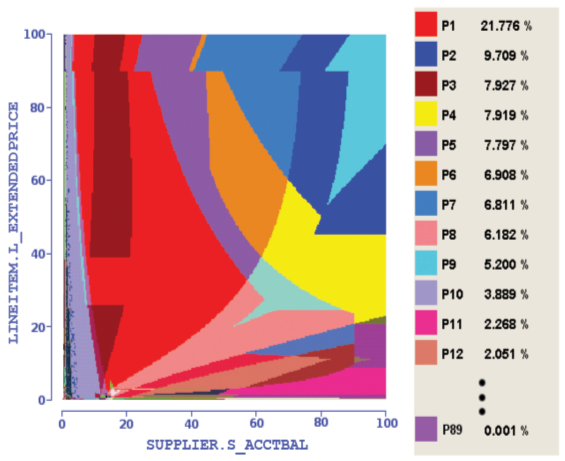
The cost of a given query execution plan is a function of many parameters, including the database structure and contents, the engine settings, the system configuration, etc. For a query on a given database and system configuration, the optimizer’s plan choice is primarily a function of the *selectivities* of the base relations participating in the query – that is, the estimated number of rows of each relation relevant to producing the final result. Varying the selectivities of one or more of the base relations produces the selectivity space w.r.t. these relations. A “parameterized query template" is a query with additional predicates that produce queries across this selectivity space.

For example, consider QT8, the parameterized 2-D query template shown in Figure 1.3, based on Query 8 of the TPC-H benchmark, with selectivity variations on the SUPPLIER and LINEITEM relations through the s_acctbal $\leq C_1$ and l_extendedprice $\leq C_2$ predicates, respectively. By varying the constants C_1 and C_2, queries are generated across the selectivity space.

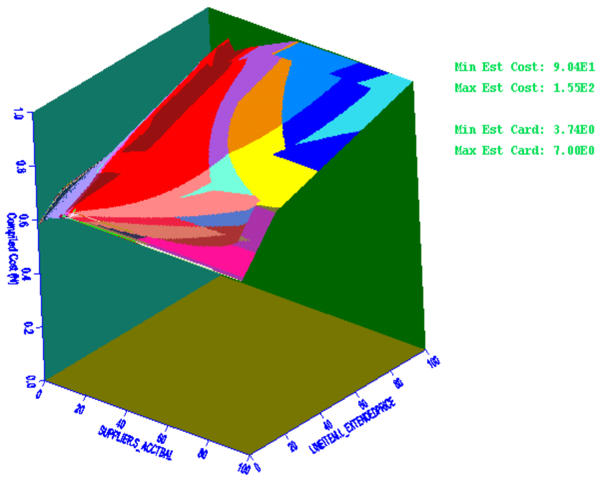
1.2 Optimizer Diagrams

The behavior of a query optimizer over the selectivity space can be captured in a suite of diagrams. First, a "plan diagram" [33] denotes a color-coded pictorial enumeration of the execution plan choices of a database query optimizer for a parameterized query template over the relational selectivity space. The plan diagram for QT8 (produced using the Picasso optimizer visualization tool on a popular commercial database engine) is shown in Figure 1.4(a), where the X and Y axes determine the percentage selectivities of the SUPPLIER and LINEITEM relations, respectively, and each color-coded region represents a particular plan that has been determined by the optimizer to be the optimal choice in that region. We find that a set of 89 different optimal plans, P1 through P89, cover the entire selectivity space. The value associated with each plan in the legend indicates the percentage area coverage of that plan in the diagram – P1, for example, covers about 22% of the space, whereas P89 is chosen in only 0.001% of the space. ***[Note to Readers: We recommend viewing all diagrams presented in this paper directly from the color PDF file, available at http://dsl.serc.iisc.ernet.in/ pooja/thesis_draft.pdf, or from a color print copy, since the greyscale version may not clearly register the various features.]***

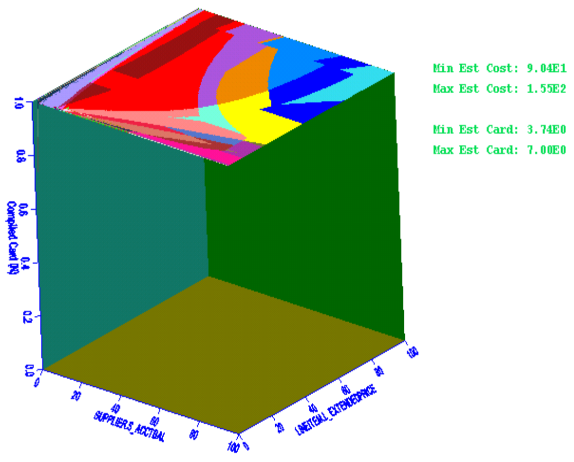
Complementary to the plan diagram is a "cost diagram", shown in Figure 1.4(b), which is a three-dimensional visualization of the estimated plan execution costs over the same relational selectivity space. The X and Y axes represent the variations in selectivity and the Z axis represents the cost. In this picture, the costs are normalized to the maximum cost over the space,



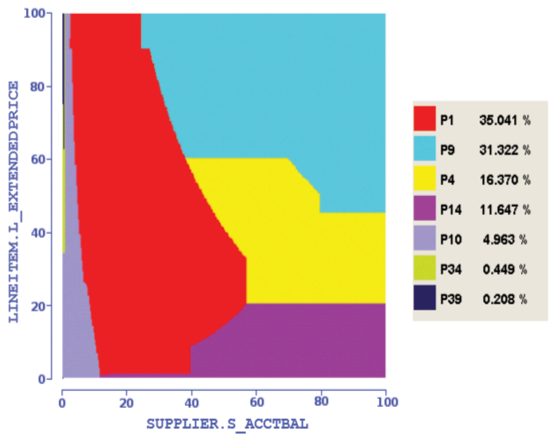
(a) **Plan Diagram**



(b) **Cost Diagram**



(c) **Cardinality Diagram**



(d) **Reduced Diagram (Threshold = 10%)**

Figure 1.4: Sample Plan and Reduced Plan Diagrams (QT8)

which in this case is 155 and occurs at the point corresponding to maximum selectivity along the X and Y axes. The minimum and maximum estimated costs are also shown in the side panel of the diagram.

Finally, a "cardinality diagram", shown in Figure 1.4(c), is similar to a cost diagram except that it shows the cardinality of the query result as estimated by the optimizer, instead of execution cost. The minimum and maximum estimated cardinalities are also shown in the side panel.

1.3 Plan Diagram Reduction

As is evident from Figure 1.4(a), plan diagrams can be extremely complex and dense, with a large number of plans covering the space – several such instances spanning a representative set of query templates based on the TPC-H benchmark, over a suite of industrial-strength optimizers, are available at [30]. An interesting research problem that now arises is whether these complex diagrams can be "reduced" to much simpler pictures featuring significantly fewer plans, *without adversely affecting the query processing quality* of any individual query point.

For example, if users were willing to tolerate a minor cost increase of at most 10% for any query point in the diagram relative to its original (optimizer-estimated) cost, Figure 1.4(a) could be reduced to that shown in Figure 1.4(d), where *only 7 plans remain* – that is, most of the original plans have been "completely swallowed" by their siblings, leading to a highly reduced plan cardinality. Further, note that a 10% increase, apart from being small in absolute terms, is also well within the bounds of the *inherent* error that characterizes the estimation process of modern optimizers [25, 35, 43]. The complete graph of the reduced plan diagram's plan cardinality as a function of the cost increase threshold for this example is shown in Figure 1.5.

The problem of reduction of plan diagrams was first posed in [4, 20]. A primary difference between their work and ours, as described in more detail in Chapter 2, is that our evaluation is on *industrial-strength* query templates and query optimizers. Their studies, on the other hand, profiled basic Select-Project-Join (SPJ) queries on home-grown optimizers. Also, we guarantee to satisfy the user-specified cost-increase threshold for every single query point, whereas they

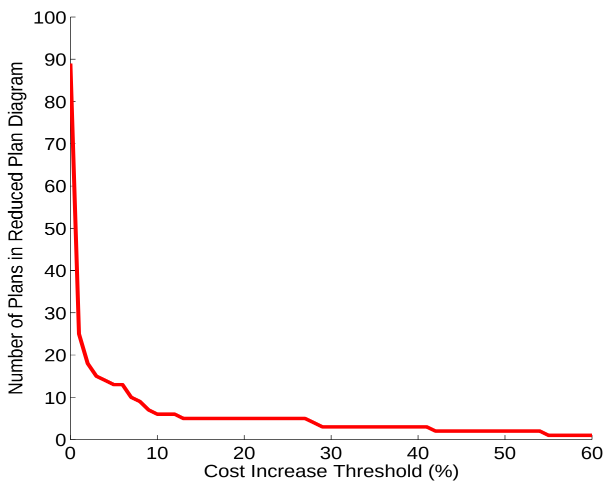


Figure 1.5: Plan Cardinality vs Cost Threshold

may fail to meet this constraint when plans have nonlinear cost functions, a common occurrence in practice.

1.3.1 Alternative Reduction Techniques

Ideally, the optimizer itself should be re-engineered to "pre-facto" (i.e. directly) produce the reduced plan diagram. This could be done by considering, for each query point, not only the optimal plan but also alternative plans that have near-optimal costs, and from this set, picking the plans so as to minimize the total number of plans over the space.

Another solution, proposed in [20], is to sample points across the parameter space and to fill the remaining space using the PQO assumptions of plan convexity, uniqueness and homogenity (See Chapter 2 for details).

However, the above solutions are not feasible in our environment for the following reasons: Firstly, we include commercial optimizers in our test-suite, and therefore the pre-facto option is inherently ruled out. Secondly, the PQO assumptions, as shown vividly in [33], do not generally hold in practice for industrial-strength database environments. Therefore, in our work, we use the more generic, albeit computationally expensive, approach of (a) exhaustively optimizing each query in the selectivity space (at a given resolution) to produce the plan diagram, and then (b) performing reduction as a "post-facto" exercise.

1.4 Benefits of Plan Diagram Reduction

We now turn our attention to highlighting a variety of benefits of performing plan diagram reduction. The reduction of plan diagrams can result in two situations:

Case 1: The number of plans in the reduced plan diagram is a *proportional* fraction of the number of plans in the original plan diagram.

Case 2: The number of plans in the reduced plan diagram is within some *absolute* number irrespective of the cardinality of the original plan diagram. An *anorexic* plan diagram is defined as a diagram whose plan cardinality is within/around a small absolute number (10 is the yardstick used in this thesis).

Although most of the benefits listed below are reaped for Cases 1 and 2, some of them become meaningful only in the case of anorexic reduced diagrams, as will be clear from the accompanying descriptions.

1.4.1 Quantification of Redundancy in Plan Search Space

Plan diagram reduction quantitatively indicates the extent to which current optimizers might perhaps be over-sophisticated in that they are “doing too good a job”, not merited by the coarseness of the underlying cost space. This opens up the possibility of redesigning and simplifying current optimizers to directly produce reduced plan diagrams, in the process lowering the significant computational overheads of query optimization. For example, a possible approach could be based on modifying the set of sub-plans expanded in each iteration of the dynamic programming algorithm to (a) include those within the cost increase threshold relative to the cheapest sub-plan, and (b) remove, using stability estimators of the plan cost function over the selectivity space, “volatile” sub-plans; the final plan choice is the most stable within-threshold plan.

1.4.2 Enhancement of PQO Usability

A rich body of literature exists on *parametric query optimization* (PQO) (e.g.[8, 12, 13, 19, 20, 23]). 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 values to identify the best plan – the expectation is that this would be much faster than optimizing the query from scratch.

A practical difficulty with PQO, however, is the representation of the plan optimality boundaries, which could, in principle, be of arbitrary complexity, making it difficult to identify specifically which plan from the set of optimal plans is to be utilized for a newly arrived query. A workaround for this problem is the following [20]: For the specific query currently supplied by the user, evaluate its estimated execution cost with *each of the plans* in the optimal set. Then, choose the lowest cost plan for executing the query. For this workaround to be viable, the plan diagram must have, in an absolute sense, only a small number of plans – this is because while plan-costing is cheap as compared to query optimization [20], the total time taken for many such costings may become comparable. However, as shown in Figure 1.4(a), the number of optimal plans can be very large, unless plan diagram reduction is applied.

Therefore, a direct benefit of plan diagram reduction is that it makes PQO viable from an implementation perspective even in the highly complex world of industrial-strength optimizers.

1.4.3 Identification of Error-Resistant Plans

Plan diagram reduction can help identify plans that provide robust performance over large regions of the selectivity space. Therefore, *errors* in the underlying database statistics, a situation often encountered by optimizers in practice [25], may have much less impact as compared to using the fine-grained plan choices of the original plan diagram, which may have poor performance at other points in the space. As is shown later in this thesis, the cost of the original plan can be orders-of magnitude greater than the replacement plan.

For example, in Figure 1.4(a), estimated selectivities of (14%,1%) leads to a choice of plan P70. However, if the actual selectivities at runtime turn out to be significantly different, say (50%,40%), using plan P70, whose cost increases steeply with selectivity, would be disastrous. In contrast, this error would have had no impact with the reduced plan diagram of Figure 1.4(d), since P1, the replacement plan choice at (14%,1%), remains the preferred plan for a large range of higher values, including (50%,40%). Quantitatively, at (50%, 40%), plan P1 has a cost of

135, while P70 is much more expensive, about *three times* this value.

In short, the final plan choices become robust to errors that cause the actual query point to lie within the optimality regions of the replacement plans. In case of errors causing the query point to lie outside this optimality region, the replacement plan can be chosen such that it provides robustness, without effecting the plan reduction process. Such robustness of plan choices is especially important for industrial workloads where often the goal is to identify plans with stable good overall performance as opposed to selecting the best local plan with potentially risky performance characteristics [27].

1.4.4 Identification of Least-Expected-Cost Plans

When faced with unknown input parameter values, today's optimizers typically approximate the distribution of the parameter values using some representative value – for example, the mean or modal value – and then always choose this "least specific cost" plan at runtime. It has been shown in [6, 7] that a better strategy would be to instead optimize for the "least expected cost" plan, where the full distribution of the input parameters is taken into account. Computing the least expected cost plan not only involves substantial computational overhead when the number of plans is large, but also assumes that the various plans being compared are all modeled at the same level of accuracy, rarely true in practice. With plan diagram reduction, on the other hand, both the efficiency and the quality of the comparisons can become substantially better since there are fewer contending plans.

1.4.5 Minimization of Overheads of Multi-Plan Approaches

A dynamic approach for selecting the best query plan was proposed in [1] wherein multiple candidate query plans are executed *in parallel*. Based on the relative rate of progress of the various plans, slower candidates are terminated along the way. The viability of this strategy is based on keeping the number of parallel candidate plans to a manageable number given the available computational resources, and plan diagram reduction can help satisfy this constraint.

An alternative and less resource-intensive multi-plan approach is proposed in [25] wherein during execution of the best compile-time plan choice, based on the observed run-time performance, a change in the query plan could be triggered for the remaining unexecuted portion of the query. A recently proposed method called "progressive query optimization" (POP) [28] alters this approach to incorporate plan validity ranges around estimated cardinalities to avoid unnecessary reoptimization that does not lead to a change of plans. POP also handles pipelined results and is therefore a more generalized approach. A further refinement of POP is the very recent RIO algorithm [3] which uses bounding boxes around cardinality estimates to represent uncertainty in these estimates, and then attempts to identify plans that retain their quality throughout the box.

When reoptimization approaches of the above kind are combined with the plan diagram reduction discussed in this thesis, the likelihood of triggering a re-optimization becomes substantially lower, thereby reducing the associated overheads.

1.4.6 Enhanced Support for Plan Clustering

Plan diagram reduction fits in perfectly with the query clustering approach previously proposed in the Plastic plan recycling tool [15, 37, 39, 50], 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. Further, from an implementation perspective, having fewer distinct plans makes it easier with regard to both storage and comparison.

1.5 Contributions

This thesis considers the problem of reducing the dense plan diagrams of modern optimizers from theoretical, statistical and empirical perspectives. We first show that finding the optimal (w.r.t. minimizing the plan cardinality) reduced plan diagram is NP-Hard through a reduction from Set Cover. This result motivates the design of CostGreedy, a greedy algorithm whose complexity is $O(nm)$, where n is the number of plans and m is the number of query points in the diagram $(n \ll m)$. Hence, for a given picture resolution, CostGreedy's performance scales *linearly* with the number of plans in the diagram, making it much more efficient than the

$O(m^2)$ reduction algorithm of [33]. Further, from the reduction quality perspective, CostGreedy provides a tight performance guarantee of $O(\ln m)$, which cannot be improved upon by any alternative deterministic algorithm.

We also consider a storage-constrained variant of the plan diagram reduction problem and find that it retains the hardness of the general problem. On the positive side, however, we provide ThresholdGreedy, a greedy algorithm that delivers a performance guarantee of 0.63 w.r.t. the optimal.

Using extremely coarse characterizations of the cost distributions of the optimal plans, we develop fast but effective estimators for determining the expected number of plans retained for a given cost increase threshold. These estimators can also be used to predict the location of the best possible tradeoff (i.e. the "knee") between the plan cardinality reduction and the cost threshold.

A detailed experimental analysis on the plan diagrams produced by industrial-strength optimizers with TPC-H and TPC-DS based multi-dimensional query templates demonstrates that:

1. Plan diagram reduction can be carried out efficiently, in just a few minutes even for extremely complex diagrams.

2. The CostGreedy algorithm typically gives the optimal reduction or is within a few plans of the optimal.

3. The analytical estimates of the plan-reduction versus cost-threshold curve are reasonably accurate and are produced in a few milliseconds.

4. Finally, and most impactfully, a *20% cost threshold* is usually amply sufficient to bring the plan cardinality to *within or around 10*, i.e. to make the reduced plan diagram anorexic, even for high dimensional query templates – this is an especially promising result from a practical perspective.

Finally, we present RobustCostGreedy, an extension of the basic CostGreedy algorithm that attempts to maintain the query processing quality of the reduced plan diagram within user specified bounds, in spite of errors that may occur in the optimizer's selectivity estimates. An experimental analysis indicates that the plans retained by RobustCostGreedy can provide orders-of-magnitude better performance than the optimizer's original choices, and are in fact often close to the optimal at the actual locations. In a nutshell, RobustCostGreedy provides "selectivity-error-resistance".

All the algorithms presented in this thesis have been implemented in the Picasso optimizer visualization tool v1.0 [30].

1.6 Organization

The remainder of this thesis is organized as follows: Related work is reviewed in Chapter 2. The problem formulation and hardness results are explained in Chapter 3 which proves NP-hardness of the plan diagram reduction problem through a reduction from Set Cover. Chapter 4 presents Algorithm CostGreedy and proves that it has a tight and optimal bound. The AmmEst estimator is explained in Chapter 5. Experimental results are highlighted in Chapter 6. An interesting application of plan diagram reduction with reference to the enhanced resistance to errors in selectivity estimates of the optimizer is described in Chapter 7. Implementation issues are discussed in Chapter 8. Finally, Chapter 9 summarizes the conclusions of our study and outlines future research avenues.

Chapter 2

Survey of Related Research

There has been extensive work in the area of query optimization for relational database management systems since the early 70's, triggered by the advent of declarative query languages. A number of surveys (eg. [18, 5]) have covered the progress of query optimization techniques over the years. We assume the reader is familiar with the techniques they discuss and only give a brief overview of the basic concepts here.

2.1 Challenges of Query Optimization

The key constituents of the query evaluation component of an SQL database system are the *query optimizer* and the *query execution engine*. The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of an SQL query as input and is responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans. One aspect of optimization is where the system attempts to find an expression equivalent to the given expression, but more efficient to execute. Another aspect is selecting a detailed strategy for processing the query. The task of an optimizer is computationally challenging since, for a given SQL query, there can be a large number of possible execution plans – specifically, for a query with n base relations, the number of plans in the strategy space is atleast $O(n!)$. Using pruning techniques such as dynamic programming, the time complexity can at best be brought down to $O(3^n)$ [41]. From an absolute time perspective, optimization overheads in the hundreds of seconds have been reported for real-world database deployments in [21]. The query execution engine implements the set of physical operators specified by the execution plan. Each operator takes as input one or more data streams and produces an output data stream. Examples of physical operators are sequential scan, index scan, (external) sort, nested-loop join and sort-merge join.

The design of a query optimizer entails tackling the following challenging issues:

2.1.1 Plan Selection Strategy

A number of selection strategies can be applied for query optimization. These include:

1. Make a random choice.
2. Use a set of heuristic rules.
3. Use randomized algorithms or genetic techniques.
4. Exhaustively enumerate the search space and use a cost-based approach.

The cost-based approach is the most commonly used in modern optimizers, since none of the others can guarantee the quality of their choices. The pioneering work in the development of cost-based optimizers was carried out in the System-R project [38]. Their techniques have been incorporated in many commercial optimizers and continue to be remarkably relevant. In System-R, the size of the search space is restricted by considering only the set of left-deep plans, which allows pipelining of the output of one operator to the input of the next operator. It also introduced the notion of "interesting orders" into the plan selection process – the ordering of the output tuples of an operator is called an interesting order if it can become useful in some subsequent operation. The idea of an interesting order was later generalized to *physical properties* [17], which refers to any characteristic of a plan that is not necessarily shared by other plans for the same expression, but could impact the cost of subsequent operations.

In an alternative strategy, Chu and Halpern [6, 7] propose the idea of picking the *least expected cost* (LEC) plan rather than the least specific cost (LSC) plan. In their first paper [6], they propose a set of algorithms to find this LEC plan and guarantee that this plan will be at least as good as the LSC plan, and typically better. They also consider parameters that could vary during the execution of the plan. They find that "*The greater the run-time variation in the values of parameters that affect the cost of the query plan, the greater the cost advantage of the LEC plan is likely to be*". They assume that the probability distribution of the values of the parameters is available at compile-time.

In their second paper [7], they observe that the LSC optimization does, in many cases, yield the LEC plan. The current optimizers can be coaxed to pick the LEC plan by appropriately choosing the parameters and their settings. They also study cases where running time is not the cost measure applied (it may matter if the plan is blocking or produces results at a constant rate, etc.) and find that in these scenarios, LEC optimization becomes particularly relevant.

2.1.2 Efficient Selection Strategies

For the cost-based optimizers, System-R proposed the use of dynamic programming to efficiently find a good plan. The dynamic programming approach is based on the assumption of the *principle of optimality* [47], which states that the optimal solution to a problem is a combination of optimal solutions to its subproblems. While dynamic programming (DP) works very well for moderately complex queries with up to around a dozen base relations, it usually fails to scale beyond this stage in current systems due to its inherent exponential space and time complexity. Therefore, DP becomes practically infeasible for complex queries with a large number of base relations.

To address the above problem, a variety of approaches have been proposed in the literature, such as Iterative Dynamic Programming (IDP) [26, 40], wherein DP is employed bottom-up until it hits its feasibility limit, and then iteratively restarted with a significantly reduced subset of the execution plans currently under consideration. A recent alternative approach that improves on IDP's performance and scalability is Skyline Dynamic Programming (SDP) [9].

When queries are optimized at the time they are submitted by the user, the selection process can add a substantial overhead to the execution time of the query. In order to avoid this, the Parametric Query Optimization (PQO) method was proposed. The goal here is to *apriori* identify the *parametric optimal set of plans* (POSP) for the entire parameter space at compile time, and subsequently to use at run time the actual parameter settings to identify the best plan – the expectation is that this would be much faster than optimizing the query from scratch. The PQO method was first proposed in [23] in the context of randomized algorithms for plan selection. They considered buffer size as the primary parameter, although their solution could work with arbitrary parameters. Subsequently, a number of PQO-based techniques have been proposed for cost-based optimizers:

In the pioneering work of Betawadkar & Ganguly [4], a System-R style optimizer with left-deep join-tree search space and linear cost models was built, the workload comprising of pure SPJ query templates with star or linear join-graphs and one-dimensional selectivity variations. They proposed the idea of finding an approximate POSP given a tolerance factor (cost increase threshold). Within this context, their experimental results indicate that, for a given cost increase threshold, plan diagram reduction is more effective with increasing join-graph complexity. They also find that "*if the increase threshold is small, a significant percentage of the plans have to be retained*". For example, with a threshold of 10%, more than 50% of the plans usually have to be retained. However, this conclusion is possibly related to the low plan cardinality (less than 20 in all the experiments) in their original plan diagrams.

In subsequent work, Hulgeri & Sudarshan [19, 20] model an optimizer along the lines of the Volcano query engine [16], and evaluate SPJ query templates with two, three and four-dimensional relational selectivities. In their first paper [19], they discuss the PQO problem in the context of linear cost functions where the conventional optimizer is unaltered. The optimizer is treated as a black-box and the plans and costs returned by the optimizer are used to find the POSP. They also propose a solution to the PQO problem in the presence of piecewise linear cost functions, which works for an arbitrary number of parameters. This solution involves altering the current optimizers to handle cost functions in the place of atomic cost values.

In the second paper [20], they propose a heuristic solution to the PQO problem which works with arbitrary nonlinear and discontinuous cost functions and any number of parameters. They propose an algorithm called *AniPQO* (and a variation of it called *DAG-AniPQO*), which requires minimal changes to existing optimizers and attempts to find a subset of the POSP such that for each point in the parameter space, either the optimal plan or a close-to-optimal plan is in the result set. The closeness to optimality is measured by an optimality threshold, which is guaranteed to be maintained in the case of linear cost functions, but cannot be guaranteed in the presence of nonlinear cost functions, when it is used only as a heuristic. Even with this relaxation, the final number of plans with a threshold of 10% can be large – for example, a 4-D query template with 134 original plans is reduced only to 53 with the DAG-AniPOSP algorithm and to 29 with AniPOSP.

Most of the solutions to the PQO problem are based on assuming cost functions that would result in one or more of the following:

1. 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;

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

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

However, it has been found that none of the three assumptions hold true, even approximately, in the plan diagrams produced by the commercial optimizers [33]. Even in situations where these assumptions hold, it is very difficult to store the regions of optimality of each of the plans in the POSP, so as to pick the best one at the time of execution. An alternative proposed by Hulgeri & Sudarshan in [19, 20], is to estimate the cost of all the plans that belong to the POSP at the time of execution and pick the one that gives the minimum cost for the actual parameter values. This will be faster than optimizing the query from scratch, provided the number of plans in the POSP is not too large.

2.1.3 Run-time Refinements of Plan Choices

Query optimizers often make poor decisions because their compile-time cost models use inaccurate estimates of various parameters. There have been several efforts in the past to address this issue, which can be categorized as – strategies that make decisions at the start of query execution and strategies that make decisions during query execution. There are some parameters, like memory availability, whose value cannot be predicted at compile-time, but are accurately known at the start of execution. Assuming that the values of these parameters remain constant for the duration of the execution, the following strategies have been proposed:

1. Perform query optimization just before query execution. This method is not very efficient, especially if the query is executed repeatedly.

2. Find the best execution plan for all possible values of the parameters and lookup the best plan for the current parameter values at runtime (PQO).

3. Perform part of the optimization at compile time and defer any decisions that are affected by the parameter values to execution time.

For parameters whose value cannot be predicted at the start of the execution, like predicate selectivities, the following strategies can be applied:

1. Antonshenkov [1] proposes a strategy where, in order to execute a query, multiple query plans are run in parallel. When one plan finishes or makes significant progress, the other competing plans are killed. This strategy assumes that ample resources are available, and is applied only to some components of the query execution (typically to individual table accesses).

2. Reoptimization:

   (a) Kabra and DeWitt [25] propose a technique which compares the expected cost and output cardinality associated with chosen operators in the plan with the corresponding runtime values. If there is a significant discrepancy, this information is used to alter the allocation of shared resources and/or reoptimize the remaining unexecuted portion of the query. Checks within a pipelined set of operators are validated at the termination of the pipeline and a change of plans is conditional to ensuring that the operations already performed are not wasted.

   (b) Markl et al [28] propose a major extension of [25] called "progressive query optimization" (POP). Here, the estimated operator cardinality is represented as a "validity range", denoting the range of ouput cardinalities over which the currently chosen (global) plan is expected to continue to be optimal. At runtime, the cardinality is dynamically compared against this range, and if it goes outside the range, reoptimization of the original query is immediately triggered. Oscillation between reoptimization and execution can occur any number of times. POP improves upon [25] in that it uses the validity ranges to ensure that reoptimization is triggered only if a change of plans is expected and not merely because there is a significant discrepancy. Also, since POP triggers reoptimization asynchronously (i.e. it does not wait for the pipeline containing the range-violating operator to complete execution), it is a more generalized approach to the problem.

(c) Babu et al [3] propose a refinement to POP called RIO, where bounding boxes are computed around cardinality estimates to represent uncertainty in the estimates. This uncertainty is determined based on the technique used to derive the estimate. Thus, a bounding box around an estimate defines the likely range of values that may actually occur at runtime. RIO then tries to identify plans whose performance within the entire bounding box is either optimal or near-optimal, so as to minimize the need for reoptimization and the loss of pipelined work.

2.2 Behavior of Industrial Strength Optimizers

Having examined the solutions to the query optimization problem proposed in the literature, we now shift focus to studying the behavior of industrial strength query optimizers in practice. In [44], Waas and Galindo devise algorithms for counting, exhaustive generation, and uniform sampling of plans from the complete search space. Using this information, they study the cost distribution of query plans. Cost distributions are of interest because they can be taken as obvious indicators of the stochastic difficulty of a problem, by simply considering the ratio of high quality to low quality plans. They find that, under the precondition that the queries are of sufficiently large size, i.e., involving more than 4 or 5 joins, the distributions obtained "*correspond to Gamma-distributions with shape parameter close to 1*". They also find that the percentage of plans that are within *twice* the optimum cost is usually around 1% of the total number of plans in the search space.

Reddy and Haritsa [33] study the behavior of industrial strength optimizers from the perspective of the optimality space, instead of the search space. They examine the variation of the plan choices across the selectivity space and find that current optimizers make extremely fine-grained plan choices. They also observe 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. 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. They show that through a process of plan diagram reduction where the query points associated with a small-sized plan are swallowed by a larger plan, it is possible to bring down the cardinality of the plan diagram to about *one-third* of the original cardinality, without materially affecting the query cost.

In this thesis, we study the problem of reducing plan diagrams (which represent the parametric optimal set of plans over the selectivity space) arising from industrial-benchmark-based query templates operating on commercial state-of-the-art query optimizers. Our results indicate that even for a small cost increase threshold, it is possible to efficiently find a small subset of the POSP that covers the entire space.

Chapter 3

The Plan Diagram Reduction Problem

In this chapter we define the Plan Diagram Reduction Problem, hereafter referred to as PlanRed, and prove that it is NP-Hard through a reduction from the classical Set Cover Problem [14]. For ease of exposition, we assume in the following discussion that the source SQL query template is 2-dimensional – the extension to higher dimensions is straightforward.

3.1 Preliminaries

The input to PlanRed is a Plan Diagram, defined as follows:

Definition 1 *Plan Diagram*

A Plan Diagram P *is a 2-dimensional* $[0, 100\%]$ *selectivity space S, represented by a grid of points where:*

1. *Each point* $q(x, y)$ *in the grid corresponds to a unique query with (percentage) selectivities* x, y *in the X and Y dimensions, respectively.*

2. *Each query point* q *in the grid is associated with an optimal plan* P_i *(as determined by the optimizer), and a cost* $c_i(q)$ *representing the estimated effort to execute* q *with plan* P_i*.*

3. *Corresponding to each plan* P_i *is a unique color* L_i*, which is used to color all the query points that are assigned to* P_i*.*

The set of all colors used in the plan diagram P is denoted by L_P. Also, we will use P_i to both denote the actual plan, as well as the set of query points for which P_i is the plan choice – the interpretation to use will be clear from the context.

With the above framework, PlanRed is defined as follows:

Definition 2 *PlanRed*

Given an input plan diagram P*, and a cost increase threshold* λ *(*$\lambda \geq 0$*), find a reduced plan diagram* R *that has minimum plan cardinality, and for every plan* P_i *in* P*,*

1. $P_i \in \mathsf{R}$*, or*
2. $\forall$ *query points* $q \in P_i$*,* $\exists P_j \in \mathsf{R}$*, such that* $\frac{c_j(q)}{c_i(q)} \leq (1 + \lambda)$

That is, find the minimum-sized "cover" of plans that is sufficient to recolor P (using only the colors in L_P) without increasing the cost of any re-colored query point (i.e. whose original plan is replaced by a sibling plan) by more than the cost increase threshold. Obviously, for $\lambda \rightarrow 0$, the reduced plan diagram will be almost identical to the original plan diagram, whereas for $\lambda \rightarrow \infty$, the reduced plan diagram will be completely covered by a single plan.

In the above definition, we need to be able to evaluate $c_j(q)$, the cost of executing query point q with the substitute choice P_j. However, this feature is not available in all database systems and is very expensive in the systems where it is available, therefore we use a bounding technique instead to limit the value of $c_j(q)$. Note that this means that the reductions we discuss here are *conservative* in that, in principle, it may be possible to reduce the diagram even more – such enhanced reductions will only further support the conclusions drawn later in this thesis.

The specific bounding technique we use is based on assuming the following:

Plan Cost Monotonicity (PCM): The cost function of each of the plans featured in the plan diagram is monotonically non-decreasing over the entire selectivity space S.

Intuitively, what the PCM condition states is that we expect the query execution cost of a plan to increase with base relation selectivities. For most query templates, this is usually the case since an increase in selectivity corresponds to processing a larger amount of input data. However, the assumption may not hold for query templates that feature negation operators such as "set difference", or short-circuit operators like "exists" – we discuss how to handle such situations below. For the remainder of this thesis, we consider only the common case of plan diagrams in which the PCM condition applies.

Based on the above, we can now state the following rule:

Definition 3 *Cost Bounding Rule*

Consider a pair of query points, $q_1(x_1, y_1)$ with optimal plan P_1 having cost $c_1(q_1)$, and $q_2(x_2, y_2)$ with optimal plan P_2 having cost $c_2(q_2)$. Then the cost of executing query q_1 with plan P_2, i.e. $c_2(q_1)$, is upper bounded by $c_2(q_2)$ if $x_2 \geq x_1, y_2 \geq y_1$.

That is, when considering the recoloring possibilities for a query point q_1, only those plan colors that appear in the *first quadrant*, relative to q_1 as the origin, should be considered. Further, if there exists a differently colored point q_2 in the first quadrant whose cost is within the λ threshold w.r.t. the optimal cost of q_1, then q_1 can be recolored with the color of q_2 without violating the query processing quality guarantee. In short, condition 2 of Definition 2 is replaced by the stronger requirement

$$\forall \text{ query points } q \in P_i, \exists P_j \in \mathsf{R}, \text{ such that } \exists r \in P_j$$
$$\text{with } r \text{ in first quadrant of } q \text{ and } \frac{c_j(r)}{c_i(q)} \leq (1+\lambda).$$

Handling non-PCM templates. As mentioned above, when a query template features negation operators (e.g "set difference") or short-circuit operators (e.g. "exists"), the PCM condition may not hold. However, as long as the template exhibits monotonicity (non-decreasing or non-increasing) along each of the selectivity axes, the above Cost Bounding Rule still applies with an appropriate choice of reduction quadrant, as shown in Table 3.1 for the 2D case.

Table 3.1: Reduction Quadrants

Cost Behavior X dimension	Cost Behavior Y dimension	Reduction Quadrant
Non-decreasing	Non-decreasing	I
Non-increasing	Non-decreasing	II
Non-increasing	Non-increasing	III
Non-decreasing	Non-increasing	IV

In the remainder of this thesis, we will characterize any plan diagram that has more than ten plans as *dense*. We use n and m to denote the number of plans and the number of query points in the plan diagram, respectively. Further, we use m_1 and m_2 to denote the diagram resolution in the X and Y axes, respectively, with $m = m_1 \times m_2$. Lastly, *BottomLeft* is used to denote the $(1, 1)$ point, *BottomRight* is used to denote the $(m_1, 1)$ point, *TopLeft* is used to denote the $(1, m_2)$ point, and *TopRight* is used to denote the point with coordinates (m_1, m_2) in the diagram.

3.2 The Set Cover Problem

We now move on to the classical Set Cover problem, defined as follows:

Definition 4 *Set Cover*

Given a finite universal set U, and a collection $S = \{S_1, S_2, \ldots S_n\}$ of subsets of U such that $\bigcup_{i=1}^{n} S_i = U$, find the minimum cardinality subset $C \subseteq S$, such that C covers U i.e. all elements of U belong to some subset in C.

Let $I = (U, S)$ denote an instance of a Set Cover problem. From a given instance I, create a new instance $I' = (U', S_{new})$ such that:

1. $S' = \{e'\}$, where e' is an element not in U
2. $U' = U \bigcup S'$, $S_{new} = S \bigcup \{S'\}$

Let C' be an optimal solution of I'. It is easy to see that $C = C' \setminus \{S'\}$ is an optimal solution of the original instance I. Therefore, we will assume henceforth in this section that the Set Cover instance is of the form I'.

Lemma 1 *Given a set cover instance I', addition of a new element e to U', to subset S', and to zero or more subsets in $\{S_1, S_2, \ldots, S_n\}$, does not change the optimal solution of I'.*

Proof: Let $C = \{S', S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ be the optimal solution of I' before the addition of the element e. After adding e to I', C still covers U', since $e \in S'$.

To see that C continues to be the optimal solution of I' after adding e, assume the contrary. Let C' be a cover for U' with $|C'| < |C|$. Remove e from all subsets in C' that contain e. Now C' covers $U' \setminus \{e\}$. This contradicts our selection of C as the optimal solution of I' before the addition of e. $\blacksquare$

3.3 Reducing Set Cover to PlanRed

We now show that the Set Cover problem can be reduced to the plan diagram reduction problem. Specifically, Algorithm Reduce in Figure 3.1 converts an instance of Set Cover to an instance of PlanRed. It takes as input the instance I' and threshold λ and outputs a plan diagram and another instance $I_{new} = (U_{new}, S'_{new})$ of Set Cover.

The data structures used in the algorithm are as follows:

1. $cur(q)$: integer denoting the smallest i such that query point $q \in S_i$ (i.e. denotes current plan that q belongs to in the plan diagram)

2. $belong(q)$: list storing all j, such that $q \in S_j$ and $j \neq cur(q)$ (denotes the set of plans that can be used instead of the current plan in the reduced plan diagram)

3. $cost(q)$: value indicating the cost of q in the plan diagram

4. $color(q)$: integer denoting the color (equivalently, plan) of q in the plan diagram

In addition, the value $n+1$ is used to denote the set S', i.e. $S_{n+1} = S'$ in cur and $belong$.

Algorithm Reduce works as follows: Consider a Set Cover instance $I' = (U', S_{new})$. For each subset $S_i \in S_{new}$, a unique color L_i which represents the plan P_i is created. Each element $q \in U'$ represents a query point in P, and let q be in subsets $S_{i_1}, S_{i_2}, \ldots S_{i_k}$ for each $S_{i_j} \in S_{new}$, $j = 1, 2, \ldots k$ and $i_1 < i_2 < \ldots < i_k$. Plan P_{i_1} is chosen as the representative for q and becomes the plan with which q is associated. For each of the other subsets in which q is present, a new query point r is created and placed to the right of q in the plan diagram, with its color corresponding to the subset it represents and its cost being $(1+\lambda)$ times the cost of q. Intuitively, this means that plan P_{i_1} can be replaced by plans $P_{i_j}, j = 2, 3 \ldots k$. Then, a query point t is

```
Reduce(Set Cover $I'$)

 1. Initialize $I_{new} = I'$; $\forall q \in U'$, set $belong(q) = NULL$
 2. For each element $q \in U'$
    (a) Let $q$ belong to sets $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$; $1 \leq i_1 < i_2 < \ldots < i_k \leq n+1$
    (b) Set $cur(q) = i_1$
    (c) Add $i_2, i_3, \ldots, i_k$ to $belong(q)$
 3. Let $m = |U'|$; $mx = \max_q(|belong(q)|)$ + 2 , $q \in U'$; $i$=1; Initialize $cost$
 4. Create $n+1$ colors $L_1, L_2, \ldots, L_{n+1}$
 5. Create an $m \times mx$ grid
 6. For each element $q \in U'$
    (a) Add $q$ at point $(i, 1)$ in the grid
    (b) Set $color(q) = cur(q); cost(q) = cost; cost = cost * (1 + \lambda); p = 2$
    (c) For each $j \in belong(q)$
        i. Create element $r$. Set $cur(r) = j$
        ii. $\forall z, z \in belong(q)$ such that $z > j$, add $z$ to $belong(r)$
        iii. Add $(n+1)$ to $belong(r)$
        iv. Add $r$ at position $(i, p)$ in the grid. $p = p + 1$
        v. Set $color(r) = j, cost(r) = cost$
        vi. Add $r$ to instance $I_{new}$ such that $r \in S_j$, if $j = cur(r)$ or $j \in belong(r)$
    (d) Create element $t$. Set $cur(t) = n + 1$, $belong(t) = NULL$
    (e) $cost = cost * (1 + \lambda)$
    (f) Add $t$ at position $(i, p)$ in the grid
    (g) Set $color(t) = n + 1; cost(t) = cost; cost = cost * (1 + \lambda)$.
    (h) Add $t$ to $I_{new}$.
    (i) Set $i = i + 1$
 7. For every empty point in the grid:
    (a) Create a new element $q$. Set $cur(q) = n + 1, belong(q) = NULL.$
    (b) Add $q$ to the empty point. Set $color(q) = n + 1$
    (c) Set $cost(q)$ = cost of row's rightmost point with color $L_{n+1}$
    (d) Add $q$ to $I_{new}$
 8. End Algorithm Reduce
```

Figure 3.1: Algorithm Reduce

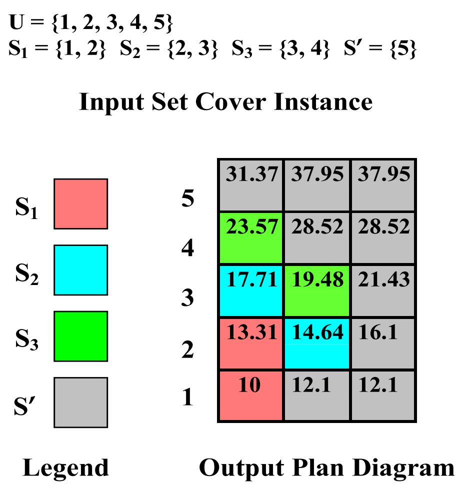


Figure 3.2: Example of Algorithm Reduce

created having plan P' corresponding to the subset S' with a cost $(1+\lambda)^2$ times the cost of q – this point is added to the right of all the points that were previously created for q. This means that t can in turn replace all the other points that were created for q, but not q itself. (Note that this process is identical to the element addition process of Lemma 1.) When moving from the last element of one row to the first element of the next row, the cost is further increased by a factor of $(1+\lambda)$.

Starting from the bottom row and moving upwards, the above procedure is repeated for each element, resulting in each element and its associated generated points being assigned to different rows in the plan diagram. Finally, for each empty point in the grid, a new query point q is created having plan P' corresponding to the subset S' with a cost equal to the cost of the rightmost point in its row with the plan P'. An example of this reduction, with $\lambda = 10\%$, is shown in Figure 3.2, where each point is represented by a square block. The blocks in the first column of the output plan diagram represent the elements originally in U, while the remaining blocks are added during the reduction process. The values in the blocks represent the costs associated with the corresponding points, and each subset is associated with a color, as shown in the legend.

We now show that Algorithm Reduce does indeed produce a plan diagram whose optimal solution gives the optimal solution to the Set Cover instance used, and hence that PlanRed is NP-Hard.

Lemma 2 *The grid G produced by Algorithm Reduce is an instance of PlanRed.*

Proof:

1. Each point in G is associated with a color (equivalently, plan) and a cost.

2. For any point (x, y) on G, where x and y represent the row and column respectively, let c = cost associated with (x, y). At point $(x, y + 1)$, the cost associated is either c or $c * (1 + \lambda)$. At point $(x + 1, y)$ the cost is greater than $c * (1 + \lambda)$ because Algorithm Reduce increases the cost by a factor of $(1 + \lambda)$ while moving from one row to the next. Therefore, the cost bounding rule of Definition 3 holds.

Hence the grid G satisfies the conditions necessary for the Plan Diagram of PlanRed. ■

Lemma 3 *The optimal solution for the instance of the plan diagram generated by Algorithm Reduce gives the optimal solution for the Set Cover instance I' used as input to the algorithm.*

Proof: Consider the plan diagram grid G and the Set Cover instance $I_{new} = (U_{new}, S'_{new})$ that is the output of the algorithm. For every point $q(x, y)$ on the grid that can be recolored, there must exist a point with that color to the right of $q(x, y)$ with cost either c or $c*(1+\lambda)$ where c is the cost of $q(x, y)$. Also, the color's index will be in the $belong$ list of the element corresponding to that point.

For each such point $q(x, y)$, there is an element r in I_{new}, such that r belongs to the subsets $S_j \in S'_{new}$, whenever $cur(q) = j$ or $j \in belong(q)$. Hence, from the above property, if point $q(x, y)$ has color L_i in the reduced plan diagram R, then the corresponding element in I_{new} will be an element of set S_i.

Therefore, if R has colors (plans) $L_R = \{L_{i_1}, L_{i_2}, \ldots, L_{i_k}\}$, since every point is colored with some color in L_R, its corresponding element in I_{new} will belong to some subset in $C_{new} = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$. Therefore, C_{new} covers U_{new}. Hence we just need to show that if L_R is the optimal color set (with least number of colors), then C_{new} is the optimal set cover for I_{new}.

To prove the above, assume the contrary, that is, that $C'_{new} = \{S_{j_1}, S_{j_2}, \ldots, S_{j_l}\}$, $l < k$ is the optimal cover of U_{new}. By construction of the grid, every point in the grid corresponding to an element in S_{j_i} $i \in \{1, 2, ...l\}$, can be colored with color L_{j_i}. Apply this color to the point in the grid and set the cost of this point to be the cost of the point with the matching color to its right. After recoloring the grid in this manner, we get a new color set $L'_R = \{L_{j_1},\ L_{j_2},\ \ldots,\ L_{j_l}\}$ that covers the whole grid with $|L'_R| < |L_R|$. This contradicts the assumption that L_R was the optimal color set. Hence, the optimal solution to the grid gives the optimal solution for the set cover instance I_{new}.

The newly created elements that are added to I' to create I_{new} by the algorithm are in accordance with Lemma 1. Hence the optimal solution for I' is the same as the optimal solution of I_{new}. Thus the optimal solution for the instance of plan diagram generated by Algorithm Reduce gives the optimal solution for the Set Cover instance I' used as its input. ■

Armed with the above lemmas, we now state the main theorem:

Theorem 1 *The Plan Diagram Reduction Problem is NP-Hard.*

Proof: It can be seen that

1. Algorithm Reduce has a polynomial time complexity of $O(nm)$.

2. For $I' = (U', S_{new})$, the grid created has in the worst case $|U'| * (|S_{new}|)$ elements with $|S_{new}|$ plans. It is a valid plan diagram. (Lemma 2)

3. The optimal solution for Set Cover Instance I' can be obtained by the optimal solution of the plan diagram generated by the algorithm. (Lemma 3)

Hence the theorem. ■

3.4 Single-swallowing PlanRed

In the hope of finding a polynomial-time optimal solution, we also considered a situation where, rather than allowing a plan to be collectively swallowed by a group of sibling plans, we mandate that a plan can be swallowed only if it can be entirely replaced by a *single* sibling plan. That is, all query points of a swallowed plan have the identical replacement color. Unfortunately, however, this constraint does not change the complexity of the problem, as proved below.

The Single-swallowing PlanRed problem is defined as follows:

Definition 5 *Single-swallowing PlanRed*

Given an input plan diagram P*, and a threshold* λ*, find the reduced plan diagram* R *with minimum plan cardinality such that for every plan* P_i *in* P*,*

1. $P_i \in \mathsf{R}$*, or*
2. $\exists P_j \in \mathsf{R}$*, such that* $\forall$ *query points* $q \in P_i, \dfrac{c_j(q)}{c_i(q)} \leq (1+\lambda)$

Applying the bounding rule of Section 3.1, the second condition is converted to the stronger requirement:

$$\exists P_j \in \mathsf{R}, \text{ such that } \forall \text{ query points } q \in P_i \; \exists r \in P_j$$
$$\text{with } r \text{ in first quadrant of } q \text{ and } \frac{c_j(r)}{c_i(q)} \leq (1+\lambda).$$

We find that enforcing the single-swallowing restriction does not change the NP-Hard complexity of the plan diagram reduction problem and show this by reducing a variation of the Dominating Set problem in a Directed Graph into an instance of Single-swallowing PlanRed. For the purpose of our reduction, we will be using an instance of the Dominating Set problem where the directed acyclic graph $G = (V, E)$ is connected and has the following structure:

1. $|V| = n + m + 1$ for some positive integers n, m
2. There is one node(root) with $indegree = 0$
3. There is a directed edge between the root and n nodes starting from the root.
4. There are a set of $k > 0$ edges between the above n nodes and the remaining m nodes starting from the set of n nodes.

Lemma 4 *The Dominating Set problem in a Directed graph with the given structure is NP-Hard.*

Proof: Let $I = (U, S)$ be a set cover instance with $|U| = m$ and $|S| = n$. Create a graph $G = (V, E)$ such that

1. For each $S_i \in S$, create a node v_i (v nodes) and for each element $e_i \in U$ create a node u_i (u nodes). Create another node w.

2. Let $V = \{u_1, u_2, \ldots u_m, v_1, v_2, \ldots v_n, w\}$

3. Let $E = \{(v_i, u_j) : e_j \in S_i\} \bigcup \{(w, v_i), \forall i = 1 \ldots n\}$

Let $D' = w, u_{i_1}, u_{i_2}, \ldots u_{i_k}, v_{j_1}, v_{j_2} \ldots v_{j_l}$ be the minimum dominating set for G. Every node u_i has a parent v_j. Hence, we can get another minimum dominating set $D = w, v_{z_1}, v_{z_2}, \ldots v_{z_k}, v_{j_1}, v_{j_2} \ldots v_{j_l}$ for G. This means that these set of v nodes has atleast one edge to all the u nodes. This implies that $C = \{S_{z_1}, S_{z_2}, \ldots S_{z_k}, S_{j_1}, S_{j_2}, \ldots S_{j_l}\}$ covers U. To see that C is the optimal cover, if there was a cover $C' = \{S_{x_1}, S_{x_2}, \ldots S_{x_h}\}$, with $|C'| < |C|$, then we can get $D'' = \{w, v_{x_1}, v_{x_2}, \ldots v_{x_h}\}$ as a minimum dominating set for G, due to the construction of G, with $|D''| < |D|$. This contradicts the assumption that D is the minimum dominating set.

Hence, we can reduce a Set Cover problem to an instance of the Dominating set problem for the directed graph structure mentioned above. Hence the Lemma. ■

We now reduce the above dominating set problem to Single-swallowing PlanRed problem.

Theorem 2 *The Single-swallowing Plan Diagram Reduction Problem is NP-Hard.*

Proof: Let $G = (V, E)$ be a directed acyclic graph having the structure mentioned earlier. Let $V = \{v_1, v_2, ...v_n\}$ and set $U = \emptyset$

1. For each node v_i create a set $S_i = \{q_i\}$ and $U = U \bigcup \{q_i\}$

2. For each edge (v_i, v_j) perform $S_i = S_i \bigcup \{q_j\}$

It can be seen that (U, S) forms an instance of the set cover problem whose optimal solution gives the optimal solution of the Directed Dominating Set problem.

This instance of the set cover problem can then be converted into a plan diagram by using the Algorithm Reduce given in Table 3.1. We make a slight modification in Algorithm Reduce, wherein, rather than choosing the set with smallest index as its representative color, we will instead choose the set with the same index as the element as its representative color. (This can be done because, while a set is created, a corresponding element is also created for it). We know by Lemma 3 that the optimal solution of the Plan Diagram formed by Algorithm Reduce gives the optimal solution of the Set Cover instance used as input to it. Also, this reduction runs in polynomial time. Hence, it will suffice for us to just show that the optimal solution to the plan diagram thus formed conforms with the aforementioned restriction.

Let $C = \{C_1, C_2, ...C_k\}$ where $C_i \in S = \{S_1, S_2, ..S_n\}$ be the optimal solution to the plan diagram reduction problem. (Recall that we represent a Plan by its corresponding set in S). Let plan $S_i \notin C$. Since the only element of S_i that is colored with color C_i in the plan diagram is x_i, it should be in some set S_j in the optimal solution. Hence, as required by the restriction, the plan S_j completely replaces S_i. ■

3.5 Storage-budgeted PlanRed

In practice, it is often the case that a fixed storage budget is provided to hold the set of plans for a query template. That is, a budget in terms of the number of stored plans, say k, is specified, and the goal is to identify the best set of k plans that would minimize the cost increase in the Reduced Plan Diagram. This problem is the *dual* of PlanRed, in terms of exchanging the constraint and the objective, and is defined as follows:

Definition 6 *Storage-budgeted PlanRed*

Given a plan diagram P *and storage constraint of retaining at most k plans, find the k plans to be chosen so as to minimize the maximum cost increase of the query points in the reduced plan diagram* R.

A Karp Reduction [14] can be used to show that Storage-budgeted PlanRed is NP-Hard by using it to solve the general plan diagram reduction problem, leading to the following theorem:

Theorem 3 *The Storage-budgeted Plan Diagram Reduction Problem is NP-Hard.*

Proof: We prove the hardness of the problem by using it to solve PlanRed. Assume that a polynomial time algorithm called $FindPlans$ gives the optimal solution for the Storage-budgeted PlanRed problem. It takes as input the number of plans and returns the set of plans so as to minimize the cost increase threshold. The method $FindThreshold$ takes as input the set of plans

```
ReducePlans($PlanDiagramP$, $threshold$)
  1. Initialize $minplans$ = All Plans in $P$
  2. for $i = n$ to 1 do
      (a) $plans = FindPlans(i)$
      (b) $th = FindThreshold(plans)$
      (c) if $threshold \leq th$
            i. return $minplans$
      (d) $minplans = plans$
  3. End Algorithm ReducePlans
```

Figure 3.3: Algorithm ReducePlans

to be retained in the reduced plan diagram and returns the minimum cost increase threshold that results.

Consider the algorithm $ReducePlans$ given in Figure 3.3, which takes as input the cost increase threshold and returns the minimum number of plans to be retained without violating this threshold. It can be seen that this algorithm runs in polynomial time as it invokes the $FindPlans$ method at most n times where n is the number of plans in the plan diagram. Thus, we have a polynomial time solution to the PlanRed Problem if we have polynomial time solution to the Storage-budgeted PlanRed Problem, which means we have a polynomial time solution to the Set Cover problem. Hence the theorem.

Chapter 4

Greedy Plan Diagram Reduction

Given the hardness results of the previous section, it is clearly infeasible to provide optimal plan diagram reduction, and therefore we now turn our attention to developing efficient greedy algorithms.

We first consider AreaGreedy, the reduction algorithm proposed in [33], where the greedy heuristic is based on plan areas. Then we present CostGreedy, a new reduction algorithm that is greedy on plan costs. Its computational efficiency and reduction quality guarantees are quantified for the general PlanRed. We then present a greedy algorithm ThresholdGreedy that has strong performance bounds for the storage-budgeted version. As before, for ease of exposition, we assume that the input plan diagram is 2-dimensional – the algorithms can be easily generalized to higher dimensions, while the theoretical results are independent of the dimensionality.

4.1 The AreaGreedy Algorithm

The AreaGreedy algorithm [33] first sorts the plans featuring in the plan diagram in ascending order of their area coverage. It then iterates through this sequence, starting with the smallest-sized plan, checking in each iteration whether the current plan can be completely swallowed by the remaining plans – if it can, then all its points are recolored using the colors of the swallower plans, and these points are added to the query sets of the swallowers.

An important point to note here is that when a plan that has already swallowed some other

```
1. Create a bucket $B_i$ for each different plan $i$ in P, and put all query points having the same plan in the corresponding bucket.

2. Create a border bucket $BB_i$ for each different plan $i$ in P. Using the Edge Detector algorithm, identify the border points of each contiguous plan region and only insert those points into the corresponding bucket.

3. Sort the buckets $B_i$ in ascending order of the areas covered by their associated plans in P. Let this sorted list be $B_1, B_2, \ldots, B_n$

4. for $i = 1$ to $n$
   (a) Swallow $(B_i)$ = true
   (b) for each point $p$ in $B_i$
   (c) for $j = 1$ to $n$ and $(j \neq i)$
       i. find, if available, a point $q$ in $BB_j$ such that $q$ is in first quadrant w.r.t $p$, $cost(q)$ is within $[100\%, (100 + \lambda)\%]$ of $cost(p)$, and $cost(q)$ is the minimum across all such qualifying points in $BB_j$
   (d) if one or more $q$ points are identified from the above step, choose the $q$ point with the lowest $cost(q)$, and mark that point $p$ can be assigned to $q$'s bucket
   (e) else Swallow $(B_i)$ = false; break
   (f) if Swallow$(B_i)$= true, move all the points in $B_i$ to their assigned replacement buckets, then delete $B_i$ and $BB_i$

5. Output all the points of P with their current plan assignments based on their assigned buckets, and use the associated coloring to form the reduced plan diagram R

6. End Algorithm AreaGreedy
```

Figure 4.1: Algorithm AreaGreedy

query points is itself considered for swallowing, then the *original* costs of the previously swallowed query points are used in computing the cost increase with the current candidate swallowers. This ensures that in the final reduced plan diagram, the cost increase of all query points is within the threshold even if these points have been subject to multiple swallowings by different plans in the iterative process.

The intuition behind the design of AreaGreedy is two-fold: First, using an area basis for the swallowing iterations is likely to reduce the number of small-sized plans. This would contribute towards plan stability as discussed in the Introduction. Second, small-sized plans tend to be found near the origin and the axes of the plan diagram [20, 32, 33] – this means that they offer more scope for swallowing since their first quadrants are big and therefore likely to have many more candidate swallower plans as compared to the larger-sized plans which occur in the higher regions of the selectivity space. The algorithm is given in Figure 4.1.

By inspection, it is obvious that AreaGreedy has a time complexity of $O(m^2)$, where m is the number of query points in the plan diagram. With respect to reduction quality, let AG denote the solution obtained by AreaGreedy, and let Opt denote the optimal solution. We now show that the approximation factor $\dfrac{|AG|}{|Opt|}$ can be no better than $0.5\sqrt{m}$.

Lemma 5 *The approximation factor* $\frac{|AG|}{|Opt|} \geq 0.5\sqrt{m}$

Proof: Construct the plan diagram as follows.

1. Initialize cost c.
2. for each $i = 2 \ldots n-1$ do
    (a) create an element of color L_1, cost c and $n-1$ elements of color L_i, cost $c \times (1+\lambda)$, and an element of color L_n, cost $c \times (1+\lambda)^2$ and add it to row $i-1$ of the grid
    (b) set $c = c \times (1+\lambda)^3$

The plan diagram created above has $m = n^2 - n - 2$ points. The AreaGreedy algorithm will output the reduced set $P_{AG} = \{P_2, P_3, \ldots, P_n\}$ while the optimal solution is $P_{Opt} = \{P_1, P_n\}$. Hence

$$\frac{|AG|}{|Opt|} = \frac{n-1}{2}$$

It can be seen that

$$\frac{\sqrt{m+1}-1}{2} < \frac{n-1}{2} < \frac{\sqrt{m}+1}{2}$$

Hence, for this plan diagram ,

$$\frac{|AG|}{|Opt|} \approx 0.5\sqrt{m}$$

Hence the Lemma. ■

4.2 The CostGreedy Algorithm

We now propose CostGreedy, a new greedy reduction algorithm, which provides significantly improved computational efficiency and approximation factor as compared to AreaGreedy.

Consider an instance of PlanRed that has an $m_1 \times m_2$ grid with n plans and $m = m_1 \times m_2$ query points. By scanning through the grid, we can populate the cur and $belong$ data structures (introduced in Section 3.3) for every point. This can be done as follows: For each query point q with plan P_i in the grid, set $cur(q)$ to be i, and add to $belong(q)$ all j such that P_j can replace q. Using this, a Set Cover instance $I = (U, S)$ can be created with $|U| = m$ and $|S| = n$. Here, U will consist of elements that correspond to all the query points and S will consists of sets corresponding to the plans in the plan diagram. The elements of each set will be the set of query points that can be associated with the plan corresponding to that set.

The following lemma shows that the reduction solution for the plan diagram can be obtained from the Set Cover instance created above.

Lemma 6 *The optimal solution of the created Set Cover instance I gives the optimal reduction solution to the plan diagram* P *that is used to create the instance.*

Proof: Let $C = \{S_{i_1}, S_{i_2}, \ldots S_{i_k}\}$ be the optimal solution of I. For each query point q in P, if it belongs to a subset $S_{i_j} \in C$, then color q with color L_{i_j}. This is a valid coloring because the element q will be in subset S_{i_j} only if q can be replaced by plan P_{i_j}. Hence, $L_R = \{S_{i_1}, S_{i_2}, \ldots S_{i_k}\}$ colors all points in the plan diagram.

To show that L_R is optimal, assume that there exists $L'_R = \{L_{i_1}, L_{i_2}, \ldots L_{i_l}\}$ which covers all plans in the plan diagram with $l < k$. The cover $C' = \{S_{i_1}, S_{i_2}, \ldots S_{i_l}\}$ is a cover of I, since if a point can be colored with $L_{i_j} \in L'_R$, then it will belong to the corresponding set S_{i_j}. Since L'_R covers all points in the plan diagram, C' covers U. This contradicts the assumption that C is the optimal cover of I. Hence the lemma. ■

Lemma 6 is explicitly used in the design of CostGreedy, shown in Figure 4.2. In Lines 1 through 6, an instance $I = \{U, S\}$ of Set Cover is created. Then, in Line 8, CostGreedy calls Algorithm Greedy Setcover, shown in Figure 4.3, which takes this input instance and outputs the cover $C \subseteq S$.

By definition, the TopRight query point in P cannot be re-colored since there are no points in its first quadrant. Therefore, its color in P has to perforce also appear in R. Hence, we remove its corresponding set from the Set Cover instance (Line 7) before applying Algorithm Greedy Setcover, and then add it to the solution at the end (Line 10).

Finally, an attractive feature of CostGreedy is that a swallowed point is recolored *only once*, in contrast to AreaGreedy where a swallowed point can be recolored multiple times.

4.2.1 Complexity Analysis

In the following theorem we show that the time complexity of CostGreedy is $O(nm)$. Since it is guaranteed that $n \leq m$, and typically $n \ll m$, this means that CostGreedy is significantly more efficient than AreaGreedy, whose complexity is $O(m^2)$. Further, it also means that for a given diagram resolution, the performance is *linear* in the number of plans in the plan diagram.

Theorem 4 *The time complexity of CostGreedy is $O(mn)$, where m and n are the number of query points and plans, respectively, in the input plan diagram* P.

Proof: Let P be an $m_1 \times m_2$ grid. While populating the $belong$ and cur lists, we maintain another two-dimensional array $mincost$ of dimension $m_1 \times n$. This array is used to store the minimum costs of the query points corresponding to each plan appearing in the partial-column located above each cell in the row above the one that is currently being processed. The initial values in $mincost$ are all ∞.

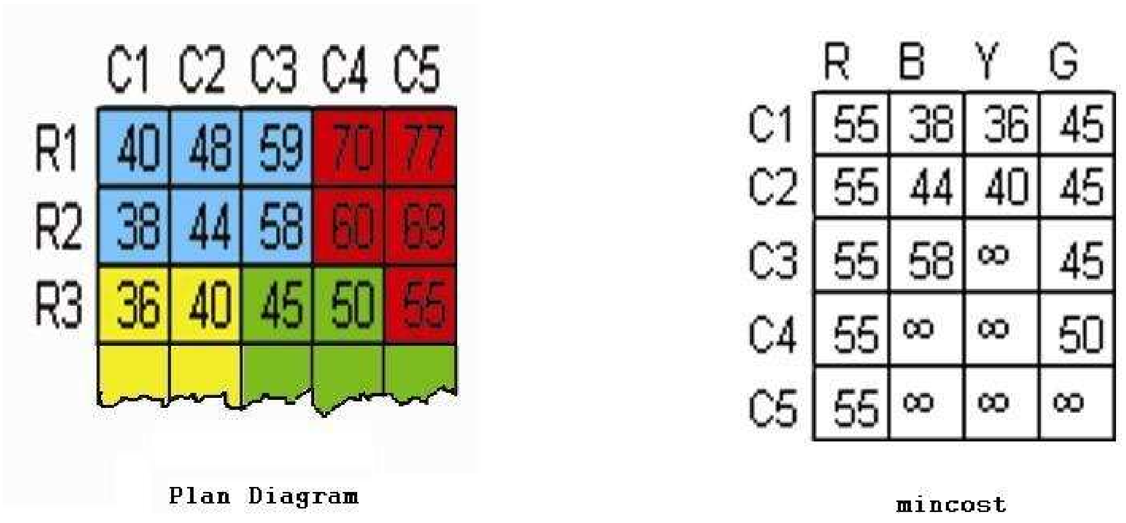
We start the scan of the grid from right to left, beginning with the top row of the grid. For each point q with plan P_k at column i in the current row, if it can be replaced by any

```
CostGreedy (Plan Diagram P, Threshold $\lambda$)
  1. For each point $q$ from $TopRight$ to $BottomLeft$ do
      (a) set $cur(q) = color(q)$
      (b) update $belong(q)$ with plans that are in $q$'s first quadrant with cost within the given threshold
  2. Let $m = m_1 \times m_2$.
  3. Create $n$ sets $S = \{S_1, S_2, \ldots S_n\}$ corresponding to the $n$ plans.
  4. Let $U = \{1, 2, \ldots m\}$ correspond to the $m$ query points.
  5. Define $\forall i = 1 \ldots n$, $S_i = \{j : i \in belong(r)$ or $i = cur(r)$ for query point $r$ corresponding to $j$,
     $\forall j = 1 \ldots m\}$
  6. Let $I = (U, S)$, $I$ be an instance of the Set Cover problem.
  7. Let $L_n$ be the color of the $TopRight$ point. Remove set $S_n$ and all its elements from $I$.
  8. Apply Algorithm Greedy Setcover to $I$. Let $C$ be the solution found.
  9. $C = C \bigcup \{S_n\}$
 10. Recolor the grid with colors corresponding to the sets in $C$ and update new costs appropriately.
     If a point belongs to more than one subset, then color it with the color that requires the least cost
     increase.
 11. End Algorithm CostGreedy
```

Figure 4.2: Algorithm CostGreedy

```
Greedy Setcover(Set Cover $I$)
  1. Set $C = \emptyset$
  2. While $U \neq \emptyset$ do:
      (a) Select set $S_j \in S$, such that $|S_j| = max(|S_i|); \forall S_i \in S$ (in case of tie, select set with smallest
          index)
      (b) $U = U \setminus S_j$, $S = S \setminus \{S_j\}$
      (c) $C = C \bigcup \{S_j\}$
  3. Return $C$
  4. End Algorithm Greedy Setcover
```

Figure 4.3: Algorithm Greedy Setcover



	C1	C2	C3	C4	C5
R1	🟦 40	🟦 48	🟦 59	🟥 70	🟥 77
R2	🟦 38	🟦 44	🟦 58	🟥 60	🟥 69
R3	🟨 36	🟨 40	🟩 45	🟩 50	🟥 55
	🟨	🟨	🟩	🟩	🟩

Plan Diagram

	R	B	Y	G
C1	55	38	36	45
C2	55	44	40	45
C3	55	58	∞	45
C4	55	∞	∞	50
C5	55	∞	∞	∞

mincost

Figure 4.4: Updating *mincost* in Algorithm CostGreedy

other plan P_j, then $mincost[i][P_j]$ should be within the increase threshold of the cost of q. Hence, through a single scan of $mincost[i]$, we can populate $belong(q)$. Then the cost of q is updated for $mincost[i][P_k]$. Since the values in the column $mincost[i]$ are candidates for the minimum values of the column $i-1$, $mincost[i-1]$ is updated with the value $min(mincost[i], mincost[i-1])$. An example is shown in Figure 4.4. The array $mincost$ contains updated values after processing all the columns of the first three rows of the plan diagram.

With the above procedure, when moving to the next row to be processed, the columns $mincost[i]$ will automatically contain the minimum costs of all the plans appearing in the first quadrant of the query point at the i^{th} column of the previous row. When a query point at column i is being processed, due to the cumulative updation of the costs of the plans visited on that row, $mincost[i]$ will be updated with the minimum costs of all the plans in that point's first quadrant. So each query point requires $2n$ iterations to be made, and there are m query points. Hence the time required for populating the data structures cur and $belong$ is of the order $O(mn)$.

Obtaining the Set Cover instance from the above data structures takes $O(mn)$ time, and the Algorithm Greedy Setcover also has a time complexity of $O(mn)$. Thus the CostGreedy has an overall time complexity of $O(mn)$. Hence the theorem. ■

4.2.2 Approximation Factor

We now quantitatively assess the approximation factor that can always be guaranteed by the CostGreedy algorithm with respect to the optimal.

Lemma 7 *CostGreedy has an approximation factor* $\frac{|CG|}{|Opt|} = O(\ln m)$*, where* m *is the number of query points in the plan diagram.*

Proof: It has been shown in [11, 42] that Algorithm Greedy Setcover (GS) has an approximation factor $\frac{|GS|}{|Opt|} \leq H(m)$, where m is the cardinality of the universal set, and $H(m)$ is the m^{th} harmonic number. The input to GS can have at most $(m-1)$ elements in its universal set (this occurs when the TopRight query point has a unique color not shared by any other point in the entire diagram). Therefore,

$$\frac{|CG|}{|Opt|} = \frac{|GS|}{|Opt|} \leq H((m-1)) = O(\ln m) \tag{4.1}$$

■

Tightness of Bound. It is shown in [42] that given any k, l where $|Greedy| = k$ and $|Opt| = l$, a Set Cover instance can be generated with $(k+l)$ sets and m elements such that $m \geq G(k,l)$, where $G(k,l)$ is a recursively defined greedy number:

$$G(l,l) = l$$

$$G(k+1,l) = \lceil \frac{l}{l-1} * G(k,l) \rceil$$

It is also shown in [42] that the following tight bound of $\ln m$ for Set Cover can be achieved using such a construction when $m = G(k,l)$:

$$\ln m - \ln\ln m - 0.31 \leq \frac{k}{l} \leq \ln m - \ln\ln m + 0.78 \tag{4.2}$$

These results are used in the following lemma.

Lemma 8 *The bound specified by Lemma 7 is tight.*

Proof: The construction process in [42] of the above-mentioned Set Cover instance, with $m = G(k,l)$, is such that every element belongs to exactly *two* sets. For a given k, l, first construct the Set Cover instance using the construction in [42]. Using this create another Set Cover instance of the form I' with $(k+l+1)$ sets and $(m+1)$ elements, as mentioned in Section 3.2. When Algorithm Reduce is applied to this new instance, it creates a grid with $m' = 3*(m+1)$ elements. This is because, for each element, since it is in two sets, it can be colored by two colors in the plan diagram. One of these will represent its current plan, and for the other plan, a new element will be created and added to its right. Then another element will be created to its right which can replace this newly created element and having the color representing the plan corresponding to the set S'. Hence, each of the $m+1$ rows will have 3 elements.

From Equation 4.2 we know that

$$\frac{|Greedy|}{|Opt|} \geq \ln m - \ln\ln m - 0.31 \tag{4.3}$$

Since $m = \dfrac{m'}{3} - 1$ it is easy to see that

$$\frac{|Greedy|}{|Opt|} = \Theta(\ln m')$$

■

Optimality of the Bound. It has been shown in [11] that the bound of $O(\ln m)$ for Set Cover is the best possible bound below which Set Cover cannot be approximated efficiently, unless NP has slightly super-polynomial-time algorithms. This result is used in the following theorem:

Theorem 5 *The bound specified by Lemma 7 is the best possible threshold below which PlanRed cannot be approximated efficiently unless NP has slightly super-polynomial-time algorithms.*

Proof: Consider a Set Cover instance $I = (U, S)$ with $|U| = m$ and $|S| = n$, where $n \leq m$. The grid G produced by Algorithm Reduce is an instance of PlanRed with $|G| = m'$, where $2m \leq m' \leq mn \leq m^2$.

Thus, $log(2m) \leq log(m') \leq 2log(m)$, which implies that a reduced bound for PlanRed will provide a reduced bound on Set Cover. But this would contradict the result of [11]. Hence, this is the optimal bound for PlanRed.

4.3 The ThresholdGreedy Algorithm

We now turn our attention to developing an efficient greedy algorithm for the Storage-budgeted variation of the PlanRed problem. Specifically, we present ThresholdGreedy, a greedy algorithm that selects plans based on maximizing the benefits obtained by choosing them. The benefit of a plan is defined to be the extent to which it decreases the cost threshold λ of the reduced plan diagram when it is chosen, which means that at each step ThresholdGreedy greedily chooses the plan whose selection minimizes the effective λ.

The least number of plans that can be in the reduced plan diagram is *a single plan* which corresponds to the plan of the TopRight query point in the plan diagram. This can be always achieved by setting the cost increase threshold λ to equal the ratio between the costs of the TopRight and BottomLeft query points in the plan diagram, i.e. $\lambda_{SinPlan} = cost(TopRight)/cost(BottomLeft)$.

We now bootstrap the selection algorithm by choosing this plan and subsequently choose additional plans based on their relative benefits. The details of the algorithm can be found in Figure 4.5. Let Ben_{opt} and Ben_{greedy} be the total benefit of choosing k plans by the optimal and greedy algorithms, respectively. This means that the final cost increase threshold with the optimal selection is $\lambda_{SinPlan} - Ben_{Opt}$, and with the threshold greedy solution is $\lambda_{SinPlan} - Ben_{TG}$. The following theorem quantifies the approximation factor of ThresholdGreedy:

Theorem 6 *Given a storage budget of k plans, let Ben_{opt} be the benefit obtained by the optimal solution's selection, and Ben_{TG} be the benefit obtained by the ThresholdGreedy algorithm's selection. Then*

$$\frac{Ben_{TG}}{Ben_{Opt}} \geq 1 - (\frac{k-1}{k})^k$$

Proof: Given that we need to choose k plans, let $TG = \{P_2, ...P_k\}$ be the plans chosen in order by the greedy algorithm. Let $Opt = \{Q_1, Q_2, ...Q_k\}$ be the plans chosen by the optimal solution. Let Ben_{P_i} and Ben_{Q_i} be the benefits of choosing the plans P_i and Q_i respectively after choosing the previous $i-1$ plans. It can be seen that

$$Ben_{TG} = \sum_{i=0}^{k} Ben_{P_i} \tag{4.4}$$

```
ThresholdGreedy ($PlanDiagramP, Budgetk$)
1. Let $P_1$ be the plan of the $TopRight$ query point.
2. Set $C = \{P_1\}$
3. $\lambda = \frac{cost(TopRight)}{cost(BottomLeft)}$
4. for $i = 2$ to $k$ do
    (a) For each plan in $P$ calculate the benefit of choosing that plan in addition to the
        plans in $C$. Let $P_j$ correspond to the plan which gives the maximum benefit.
    (b) Let $Ben$ correspond to the benefit provided by $P_j$
    (c) Set $C = C \bigcup \{P_j\}$
    (d) Set $\lambda = \lambda - Ben$
5. Recolor the grid with colors corresponding to the sets in $C$ and update new costs
   appropriately. If a point can be colored with more than one color, then color it with
   the color that requires the least cost increase.
6. End Algorithm ThresholdGreedy
```

Figure 4.5: Algorithm ThresholdGreedy

$$Ben_{opt} = \sum_{i=0}^{k} Ben_{Q_i} \tag{4.5}$$

Define B_{ij} to be the sum over all plans in P of the amount of the benefit Ben_{Q_i} that is attributed to P_j . An inequality that holds for each j is

$$\sum_{i=1}^{k} B_{ij} \leq Ben_{P_j}$$

Since P_2 is chosen first, it can be seen that

$$\forall i, Ben_{Q_i} \leq Ben_{P_2}$$

This is true because if there was some $Ben_{Q_i} > Ben_{P_2}$, then Q_i would have been chosen by the algorithm instead of P_2.

Similarly for P_3 the following inequality can be formed.

$$\forall i, Ben_{Q_i} - B_{i1} \leq Ben_{P_3}.$$

This inequality holds because, plan Q_i competes with other plans when selecting the second plan with its initial benefit Ben_{Q_i} minus the benefit that was covered by P_2.

In general these inequalities can be written as

$$\forall i, Ben_{Q_i} - B_{i1} - B_{i2}... - B_{ij-1} \leq Ben_{P_j}.$$

Adding the above set of equations over all i and using (4.4) and (4.5) we obtain the following set of k inequalities.

$$Ben_{opt} \leq k.Ben_{P_2}$$

$$Ben_{opt} \leq k.Ben_{P_3} + Ben_{P_2}$$

$$Ben_{opt} \leq k.Ben_{P_4} + Ben_{P_3} + Ben_{P_2}$$

$$\ldots$$

$$Ben_{opt} \leq k.Ben_{P_k} + Ben_{P_{k-1}} + Ben_{P_{k-2}} + Ben_{P_{k-3}}... + Ben_{P_2}$$

For a fixed Ben_{TG} the tightest bound on Ben_{opt} occurs when all of the right side in the above set of inequalities are equal, in which case we get $Ben_{P_i} = \frac{k}{k-1}Ben_{P_{i+1}}$. Using this we get

$$Ben_{TG} = \sum_{i=1}^{k}(\frac{k}{k-1})^{i-1}Ben_{P_k}$$

$$Ben_{opt} \leq k(\frac{k}{k-1})^{k-1}Ben_{P_k}$$

Using the above two equations we get

$$\frac{Ben_{TG}}{Ben_{opt}} \geq 1 - (\frac{k-1}{k})^k$$

■

For $k = 10$, which we consider to be a reasonable budget in practice, the above ratio works out to about 0.65, while for $k \rightarrow \infty$, the ratio asymptotically goes down to 0.63. In an overall sense, this means that ThresholdGreedy is always guaranteed to provide close to *two-thirds of the optimal benefit*.

Chapter 5

Estimators for Plan Diagram Reduction

Our experience has been that CostGreedy takes about a minute to carry out a single reduction on plan diagrams that have in the order of a million query points. While this appears sufficiently fast, it is likely that users may need to iteratively try out several reductions with different cost increase thresholds in order to identify the one appropriate for their purpose. For example, the user may wish to identify the "knee" of the tradeoff between plan cardinality reduction and the cost threshold – that is, the location which gives the *maximum reduction with minimum threshold*.

In the above situations, using the CostGreedy method repeatedly to find the desired setting may prove to be extremely cumbersome and slow. Therefore, it would be helpful to design fast but accurate estimators that would allow users to quickly narrow down their focus to the interesting range of threshold values. In the remainder of this section, we present such estimators.

Our first estimator, **AvgEst** , takes as input the plan diagram P and a cost increase threshold λ, and returns the estimated number of plans in the reduced plan diagram R obtained with that threshold. It uses the average of the costs of all the query points associated with a plan, to summarize the plan's cost distribution. All these averages can be simultaneously computed with a single scan of the Plan Diagram. AvgEst then sets up an instance of Set Cover, as shown in Figure 5.1, with the number of elements equal to the number of plans, and the set memberships of plans is based on their representative average costs satisfying the λ threshold. On this instance, the Greedy Set Cover algorithm, introduced earlier in Figure 4.3, is executed.

The cardinality of the solution is returned as an estimate of the number of plans that will feature in R.

```
AvgEst (Plan Diagram P, Threshold $\lambda$)
  1. Let $Cost(i)$,$\forall i = 1 \dots n$ denote the average cost of Plan $P_i$
  2. Set $U = \{1, 2, \dots n\}$
  3. Set $S_i = \{1, 2, \dots n\}$, $\forall i = 1 \dots n$
  4. for each plan $P_i$ do
       (a) For all plans $P_j$ such that $Cost(j) < Cost(i)$ or $Cost(j)$ is not within the threshold of $Cost(i)$, set $S_j = S_j \setminus \{i\}$
  5. Apply Algorithm Greedy Setcover to $I$. Let $C$ be the solution found
  6. return $|C|$
  7. End Algorithm AvgEst
```

Figure 5.1: Algorithm AvgEst

Our second estimator, **AmmEst** , uses in addition to the average value, the minimum and maximum cost values of the query points associated with a plan. That is, each plan is effectively represented by a vector of three values. Subsequently, the algorithm is identical to AvgEst, the only change being that the check for set membership of a plan is based on not just the average value but on all three representative values (min, max and average) individually satisfying the membership criterion. This algorithm is shown in Figure 5.2.

By iteratively running the estimator for various cost thresholds, we can quickly plot a graph of plan cardinality against threshold, and the knee of this curve can be used as the estimated knee. Our measurements show that this estimation process executes in a few milliseconds, orders of magnitude faster than calculating the knee using CostGreedy. Further, this estimate can be used as a starting point to find the actual knee which is likely to be in the neighborhood, as shown in the experimental results of the following chapter.

```
AmmEst (Plan Diagram P, Threshold $\lambda$)
  1. Let $Cost(i)$,$Min(i)$ and $Max(i)$,$\forall i = 1 \dots n$ denote the average, minimum and maximum cost respectively of Plan $P_i$
  2. Set $U = \{1, 2, \dots n\}$
  3. Set $S_i = \{1, 2, \dots n\}, \forall i = 1 \dots n$
  4. for each plan $P_i$ do
      (a) For all plans $P_j$ such that at least one of $Cost(j)$, $Min(j)$ and $Max(j)$ is not within the threshold of $Cost(i), Min(i)$ and $Max(i)$ respectively, set $S_j = S_j \setminus \{i\}$
  5. Apply Algorithm Greedy Setcover to $I$. Let $C$ be the solution found
  6. return $|C|$
  7. End Algorithm AmmEst
```

Figure 5.2: Algorithm AmmEst

Chapter 6

Experimental Results

Having considered the theoretical and statistical aspects of plan diagram reduction in the previous chapters, we now move on to presenting our experimental results. The testbed is the Picasso optimizer visualization tool [30], executing on a Sun Ultra 20 workstation equipped with an Opteron Dual Core 4GHz processor, 4 GB of main memory and 240 GB of hard disk, running the Windows XP Pro operating system. Through the GUI of the Picasso tool, users can submit a query template, the grid resolution and distribution at which the instances of this template should be spread across the selectivity space, the parameterized relations (axes) and their attributes on which the diagrams should be constructed, and the choice of query optimizer. With this information, the tool automatically generates the associated SQL queries, submits them to the optimizer to generate the plans, and finally produces the color-coded plan, cost and cardinality diagrams.

We conducted our plan diagram reduction experiments over dense plan diagrams produced from a variety of multi-dimensional TPC-H and TPC-DS based query templates evaluated over a suite of industrial-strength database query optimizers. The templates were instantiated at a variety of grid resolutions, based on the experimental objectives and ensuring viable diagram production times. We also confirmed that all the plan diagrams were in compliance with the plan cost monotonicity condition, described in Section 3.1. The detailed listing of the query templates used in this thesis are given in the Appendix – the naming convention used is QTx for the TPCH-based templates and DSQTx for the TPC-DS based templates, where x represents the benchmark query number on which the template is based.

A gigabyte-sized database was created using the TPC-H benchmark's synthetic generator – while the benchmark models only uniformly distributed data, we extended the generator to also produce skewed data distributions. The optimizers were all operated at their default optimization levels and resource settings. To support the making of informed plan choices, commands were issued to collect statistics on all the attributes featuring in the query templates, and the plan selections were determined using the "explain" feature of the optimizers. It is important to note here that in all our experiments, the optimizers are treated as "black boxes" and there is no attempt to customize or fine-tune their behavior. The optimizers that we use include IBM DB2 v8, Oracle 10g and Microsoft SQL Server 2005, which (due to legal restrictions) are randomly referred to as OptA, OptB and OptC in the remainder of this thesis.

6.1 Computational Efficiency

We start off by first quantitatively evaluating the runtimes of the two greedy algorithms, AreaGreedy [33] and CostGreedy (proposed in this thesis), as compared to the time taken to produce the computationally-hard optimal solution. The reduction quality of the algorithms is compared in the next section. A sample set of results on OptC is shown in Table 6.1 for QT8, the query template shown in Chapter 1, instantiated at a grid resolution of 100 uniformly distributed points per dimension[1] and reduction carried out at a cost increase threshold of 10%. We see here that even for this relatively coarse-grained situation, the optimal algorithm takes several hours to complete. In contrast, AreaGreedy takes only a few seconds, while CostGreedy is an order-of-magnitude better than AreaGreedy, finishing in a small fraction of a second.

The substantial improvement of CostGreedy with regard to AreaGreedy is, as per the discussion in Chapter 4, due to its $O(nm)$ complexity being significantly lower than the $O(m^2)$ of AreaGreedy, as $n \ll m$ in practice (recall that n is the number of plans and m is the total number of query points in the plan diagram).

[1]The QT8 plan diagram in the Introduction was obtained with a resolution of 300, resulting in a higher plan cardinality.

Table 6.1: Computational Efficiency (QT8, Res=100)

Algorithm	Original Plans	Reduced (λ = 10%)	Time
OptRed	50	7	4 hours
AreaGreedy	50	7	2.8 sec
CostGreedy	50	7	0.1 sec

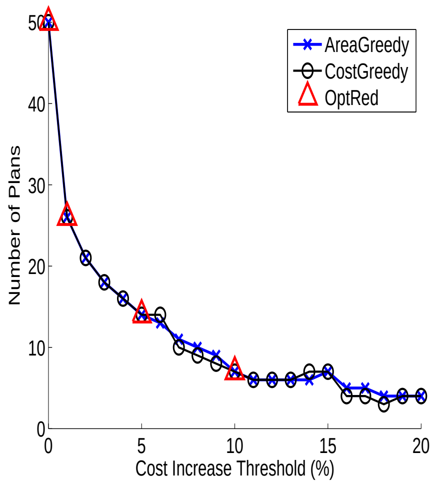


Figure 6.1: Reduction Quality (QT8), Res=100

6.2 Plan Diagram Reduction Quality

Turning our attention to the reduction quality, we see in Table 6.1 that AreaGreedy and Cost-Greedy are identical to the optimal (OptRed), all three producing reduced plan diagrams with 7 plans (in fact, the plans themselves are also the same in this case). The closeness to the optimal holds across the entire operational range of cost increase thresholds, as shown in Figure 6.1, which presents the reduced plan cardinalities for the three algorithms as a function of the threshold – only a few representative points were obtained for OptRed due to its extremely high computational overheads.

Another point to note in Figure 6.1 is the initial steep decrease in the number of plans with increasing threshold – we have found this to be a staple feature of all the dense plan diagrams that we have investigated, irrespective of the specific query template, data or query point distribution, memory availability, or database optimizer that produced the dense diagram. These settings may determine *whether or not* a dense plan diagram is produced, but if produced,

TPC-H Query Number	**OptA Plan Card**	**OptA Reduced Plans (λ=10%)**	**OptA Reduced Plans (λ=20%)**	**OptB Plan Card**	**OptB Reduced Plans (λ=10%)**	**OptB Reduced Plans (λ=20%)**	**OptC Plan Card**	**OptC Reduced Plans (λ=10%)**	**OptC Reduced Plans (λ=20%)**
2	14	7	7	20	10	8	43	12	8
5	11	4	2	12	4	4	23	6	5
8	36	4	3	16	4	2	50	7	4
9	39	9	6	18	7	3	38	4	3
10	18	5	4	7	3	3	17	4	3

Table 6.2: Plan Diagram Reduction Quality (Res = 100)

TPC-H Query Number	**OptA Plan Card**	**OptA Reduced Plans (λ=10%)**	**OptA Reduced Plans (λ=20%)**	**OptB Plan Card**	**OptB Reduced Plans (λ=10%)**	**OptB Reduced Plans (λ=20%)**	**OptC Plan Card**	**OptC Reduced Plans (λ=10%)**	**OptC Reduced Plans (λ=20%)**
2	12	11	7	23	7	6	52	14	10
5	11	4	2	11	4	3	12	5	2
8	35	5	3	24	4	2	34	6	5
9	49	10	5	34	6	5	46	3	3
10	22	7	7	12	5	4	11	2	2

Table 6.3: Skewed Data Distribution (Res = 100)

subsequently the reduction process produces consistent results. This trend is clearly seen in Table 6.2, which captures the reduction behavior of Optimizers A, B and C, with various TPCH-based query templates on which they produced dense plan diagrams.

6.2.1 Skewed Data Distribution

The above results were obtained with uniformly distributed data generated using the TPC-H benchmark's synthetic generator. We extended the generator to also produce skewed data distributions. When this skewed data was used instead, the observed reduction results did not materially change. While the specific plan diagram changed, the reduction behavior continued to be as before. This can be seen in Table 6.3, which captures the behavior of the three optimizers on their dense plan diagrams with skewed data.

TPC-H Query Number	**OptA Plan Card**	**OptA Reduced Plans (λ=10%)**	**OptA Reduced Plans (λ=20%)**	**OptB Plan Card**	**OptB Reduced Plans (λ=10%)**	**OptB Reduced Plans (λ=20%)**	**OptC Plan Card**	**OptC Reduced Plans (λ=10%)**	**OptC Reduced Plans (λ=20%)**
2	26	12	10	25	12	10	94	26	16
5	41	8	5	18	5	5	74	10	6
8	50	6	3	19	5	3	174	7	5
9	111	12	7	21	9	4	225	18	8
10	37	7	5	11	5	4	56	6	4

Table 6.4: Exponential Query Point Distribution (Res = 100)

6.2.2 Exponential Distribution of Query Points

In the above diagrams, which were produced with a uniform distribution of query points across the selectivity space, we observed that in most cases, the density of plans is greater in the regions near the axes, that is, at low selectivity values of the base relations. This motivated us to alter the arrangement of query points to be exponentially distributed with a higher density in the low selectivity region. As expected, this led to a substantial increase in the cardinality of the original plan diagram. Despite this, we see that the reduction process remains materially unaffected. This is highlighted in Table 6.4, where we see that the plan cardinality of the reduced plan diagram decreases sharply at a low cost increase threshold, irrespective of the number of plans in the original plan diagram. For example, the plan diagram cardinality increased from 38 to 225 for QT9 on OptC, but the reduced plan diagram cardinality (with $\lambda = 20\%$) went from 3 plans to only 8 plans.

6.2.3 Increased Grid Resolution

While increasing the grid resolution may increase the number of plans in the original plan diagram (due to the unearthing of new small-sized plans between the ones found at coarser resolutions), virtually all of these new plans are swallowed at a low threshold itself. This follows from the fact that these plans, being optimal over a small region, tend to have costs close to those of their neighbors and are therefore likely to be easily swallowed.

This is clearly seen in Table 6.5, which captures the reduction behavior of the three opti-

TPC-H Query Number	OptA Plan Card	OptA Reduced Plans (λ=10%)	OptA Reduced Plans (λ=20%)	OptB Plan Card	OptB Reduced Plans (λ=10%)	OptB Reduced Plans (λ=20%)	OptC Plan Card	OptC Reduced Plans (λ=10%)	OptC Reduced Plans (λ=20%)
2	23	9	8	23	12	10	76	20	12
5	18	5	3	14	5	5	31	10	6
8	47	3	3	17	5	2	89	6	6
9	64	10	6	20	8	4	91	9	4
10	25	7	4	8	4	3	31	6	4

Table 6.5: Increased Grid Resolution (Res = 300)

mizers with the TPCH-based query templates at a grid resolution of 300 uniformly distributed query points per dimension. For example, although the plan diagram cardinality went up from 38 to 91 in case of QT9 on OptC, the reduced plan diagram cardinality (with $\lambda = 20\%$) went from 3 plans to only 4 plans. This means that for practical threshold settings, the final plan cardinality in the reduced diagram is essentially “scale-free” with regard to resolution.

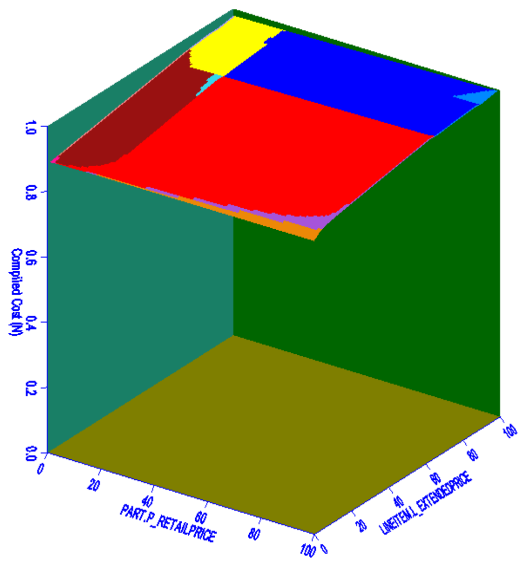
6.2.4 Reduction For Various Cost Diagram Behaviors

The behavior of the cost diagram can affect the plan diagram reduction process as it directly impacts the possibility of finding a replacement plan within the λ threshold for an entire plan. Cost diagrams can be categorized as slowly increasing or steeply increasing depending on the slope. Plan diagrams, in turn, can be categorized as sparse or dense depending on the number of plans in the diagram. Thus, the possible combinations are:

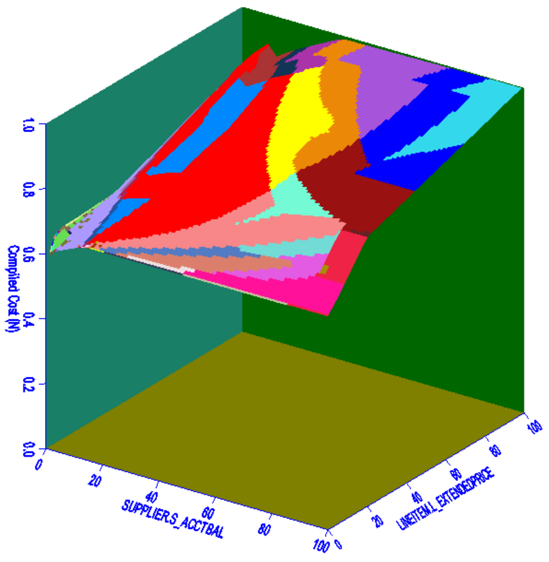
1. Slowly increasing and sparse: Since the plan diagram is already sparse, i.e. the plan cardinality is already small, there is no need for plan diagram reduction. An example of this combination, generated with QT17 on OptC, can be seen in Figure 6.2(a).

2. Steeply increasing and sparse: Since the plan diagram is already sparse, i.e. the plan cardinality is already small, there is no need for plan diagram reduction. An example of this combination, generated with QT7 on OptC, can be seen in Figure 6.2(c).

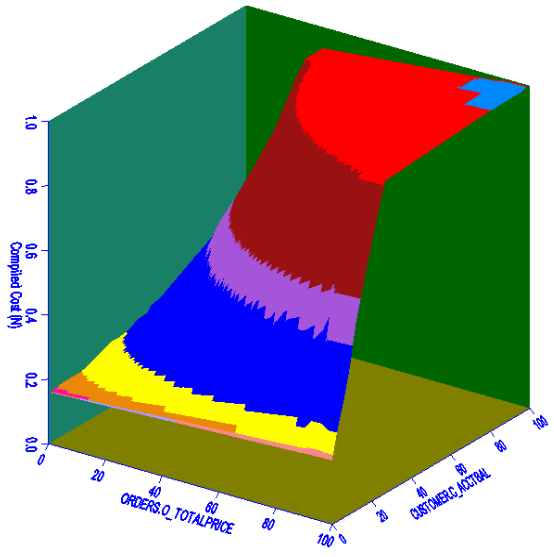
3. Slowly increasing and dense: Since the dense plan region corresponds to a slowly increasing cost region, this means that the costs of the plans in the dense region are all



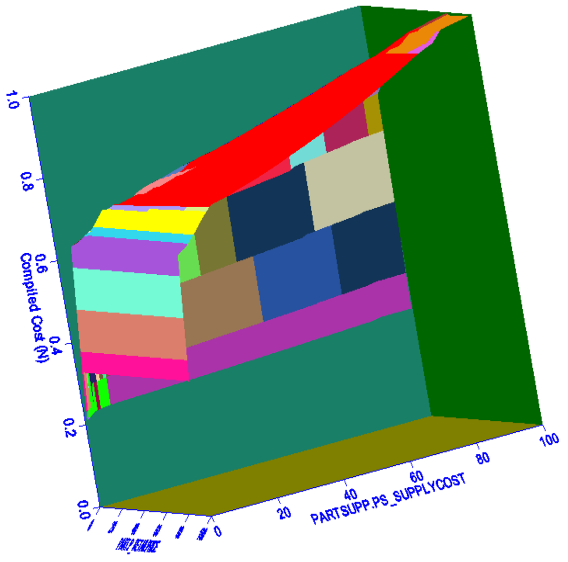
(a) **Slowly increasing and sparse**



(b) **Slowly increasing and dense**



(c) **Steeply increasing and sparse**



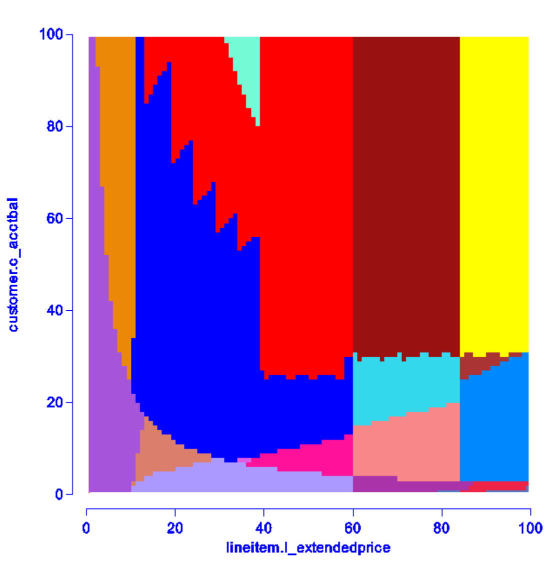
(d) **Steeply increasing and dense**

Figure 6.2: Cost Diagram Behavior

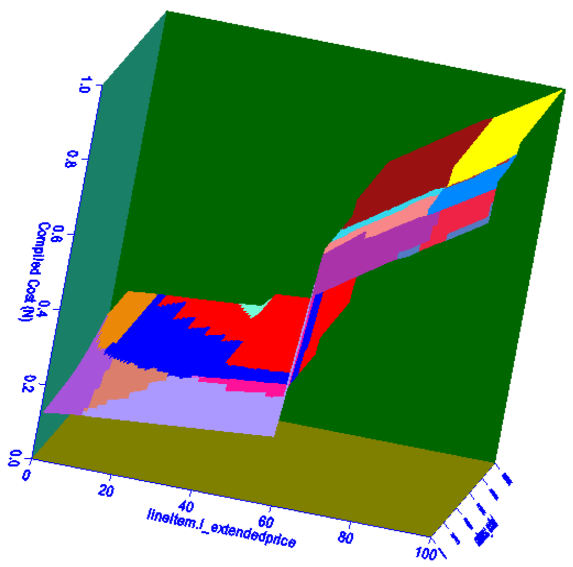
very close to each other and hence they can be swallowed at low cost increase thresholds, leading to a significant reduction in plan cardinality. A sample cost diagram with such behavior, generated with QT8 on OptC, is shown in Figure 6.2(b), where the cardinality of the original plan diagram is 50 and reduces to 4 with a cost increase threshold of 20%.

4. Steeply increasing and dense: This combination is not very commonly found. From the perspective of plan diagram reduction, this is a problem situation as the steepness of the cost function could hinder reduction at low cost increase thresholds. An example of this, generated with QT2 on OptC, is shown in Figure 6.2(d) where the steep cost regions, corresponding to low selectivity values on either base relation, have a large number of plans. The cardinality of the original plan diagram is 43 and reduces to 8 with a cost increase threshold of 20%. We see that the plan diagram reduction in the steep regions happens primarily because the plans within the steeply increasing region swallow each other.

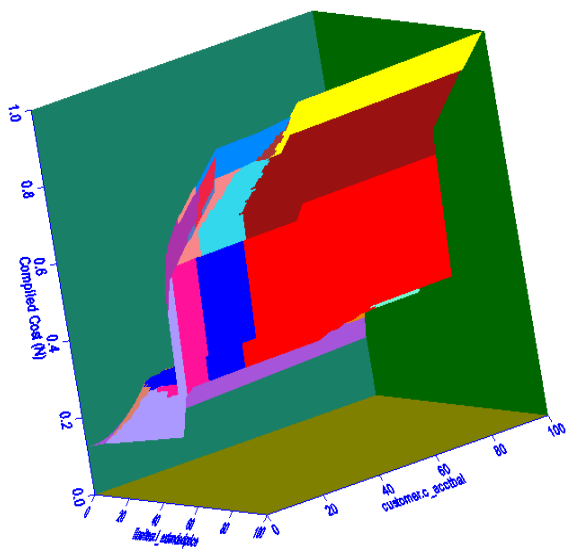
   An extreme case of the steeply increasing cost diagram and dense plan diagram would be when the cost function is *discontinuous* at any region in the diagram. This would lead to a sudden, sharp variation in the cost. A sample plan diagram of this case, generated with QT18 on an alternative optimizer, OptD, (as it was not found to occur with Optimizers A, B or C), having 18 plans is shown in Figure 6.3(a). Its corresponding cost diagram is shown in Figures 6.3(b) and 6.3(c) (from different 3-D perspectives), where the discontinuity is visible at the plane corresponding to about 60% selectivity of the LINEITEM relation. The reduced plan diagram for the same with a cost increase threshold of 20% is shown in Figure 6.3(d), where we note that the reduction still results in a small number of plans (seven) as the swallowing now happens between *neighboring* plans in the steep region of the cost diagram, in addition to the reduction in the slowly increasing cost regions.



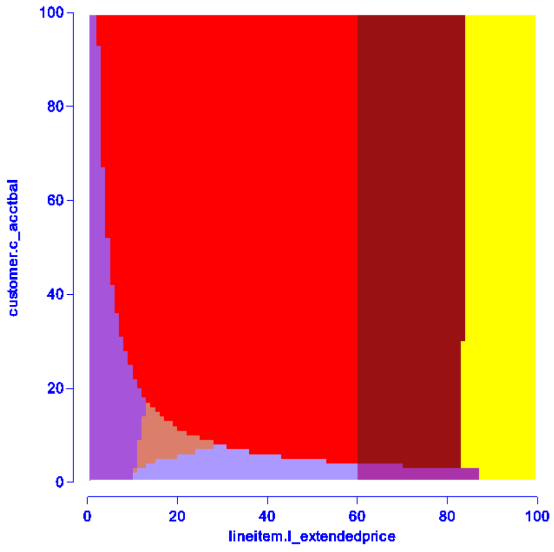
(a) **Plan Diagram**



(b) **Cost Diagram**



(c) **Cost Diagram**



(d) **Reduced Plan Diagram ($\lambda = 20\%$)**

Figure 6.3: Discontinuous Cost Function Example (QT18)

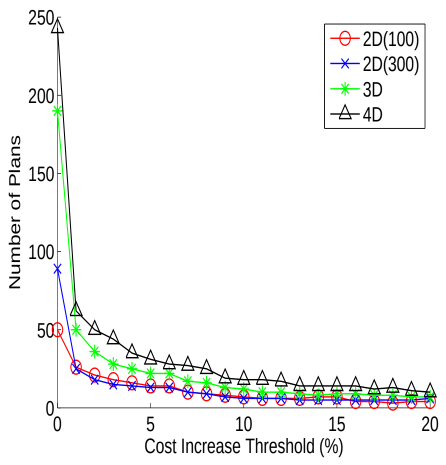


Figure 6.4: Scaling with Dimensions

6.3 Scaling with Dimensions

The above results were obtained on 2-D query templates, and we now move on to evaluating the effect of increased template dimensionality. Specifically, evaluating the behavior with 3-D and 4-D versions of the QT8 template (created through the addition of predicates c_acctbal :varies and o_totalprice :varies). This experiment was carried out only with OptC as a representative, due to the computational effort involved in producing these plan diagrams.

The results are shown in Figure 6.4 for 2-D with resolutions of 100 and 300 query points per dimension, 3-D with a resolution of 100 query points per dimension, and 4-D with a resolution of 30 query points per dimension. We see here that while the number of plans in the original plan diagram goes up steeply with increasing dimensionality, the reduction behavior is qualitatively similar across all the templates. Further, as shown in Table 6.6, the reduction behavior is remarkably stable: First, the location of the knee of the plan cardinality vs. cost increase threshold graph varies only marginally, occurring in the neighborhood of 10%. Second, the threshold required to bring the reduced plan diagram cardinality down to 10 plans is within 20%, a very practical value from a user perspective, even in a 4-D setting. Again, this seems to suggest that for practical threshold settings, the final plan cardinality in the reduced plan diagram is essentially "scale-free" with regard to dimension.

Table 6.6: Multi-dimensional Query Templates

Dim-ension	Original Plans	Knee Cost Threshold	Knee Plans	10-plan Cost Threshold
2(100)	50	8%	9	7%
2(300)	89	9%	7	7%
3	190	11%	10	11%
4	243	13%	14	20%

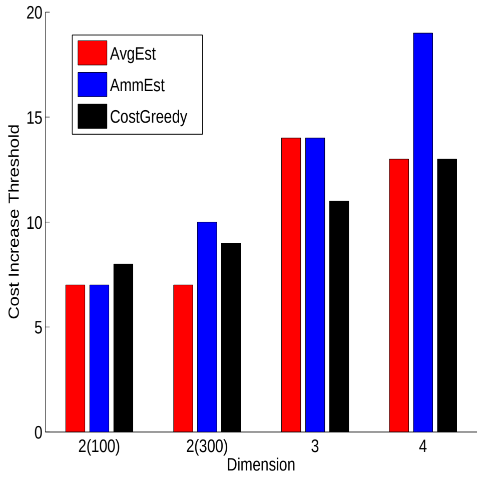


Figure 6.5: Knee Estimates

6.4 Estimator Performance

Our next experiment studies the quality of the *knee estimates* provided by the estimators. The results are shown in Figure 6.5 for QT8 on OptC (the results for other query templates and database engines are similar in nature) and indicate that AvgEst and AmmEst are reasonably accurate despite using extremely coarse characterizations of the cost distributions of plans in their optimality regions. Further, their orders-of-magnitude runtime efficiency relative to the CostGreedy algorithm, for iteratively computing the knee, is captured in Table 6.7.

The estimator performance in characterizing the full plot of reduced plan cardinality versus λ is shown in Figures 6.6(a)–6.6(d) for 2D-100, 2D-300, 3D-100 and 4D-30, respectively, the CostGreedy performance being used as the yardstick. We see here that, in general, the simple AvgEst estimator provides estimates that are closer to CostGreedy than AmmEst– however, an advantage of AmmEst is that it produces *conservative* estimates, whereas AvgEst can on occasion slightly overestimate the degree of plan diagram reduction, as is seen in Figures 6.6(a) and 6.6(b).

Table 6.7: Running Time of Estimators vs CostGreedy

TPC-H Query Template	Estimator Time(ms) (for Knee)	CostGreedy time(ms) (for Knee)
2	25	2733
5	8	1675
8	26	3648
9	71	2382
10	12	546

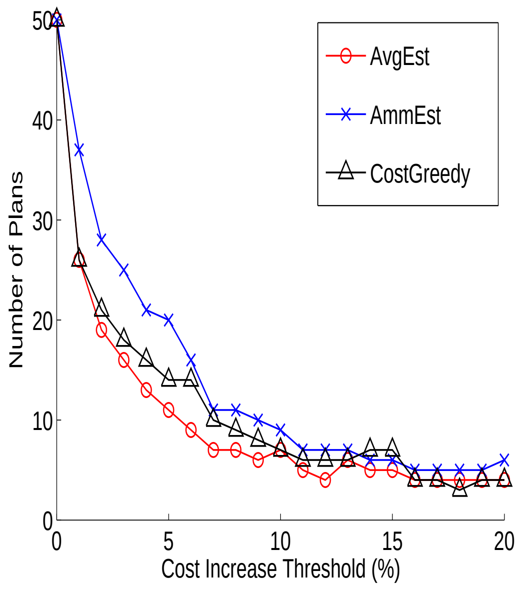
6.5 Effect of Memory Availability

In all the above results, the query parameterization was on the selectivities of the base relations. Another parameter that is well-known to have significant impact on plan choices is the amount of system memory available for query processing (e.g. Nested Loop joins may be favored in low-memory environments, whereas Hash Joins may be a more attractive alternative in memory-rich situations). In fact, plan costs can be highly non-linear or even *discontinuous* at low memory availabilities [6, 7].

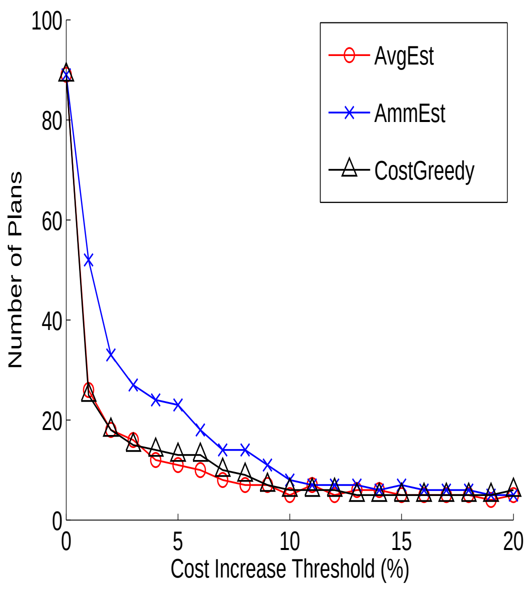
We conducted experiments wherein the memory was varied from the default system memory to the minimum permitted by the engine. The procedure to change the memory settings is given in Chapter 8. We found that the memory budget certainly had significant impact on the spatial layouts and cardinalities of the plan diagrams. For instance, with QT2 on OptA, the plan cardinality varied between 21 and 37 with varying memory for the buffer pages and the sort heap, as shown in Table 6.8. However, the basic observation that dense plan diagrams can be reduced to a few plans with low cost increase thresholds remained unchanged as shown in the

Table 6.8: OptA- Varying memory

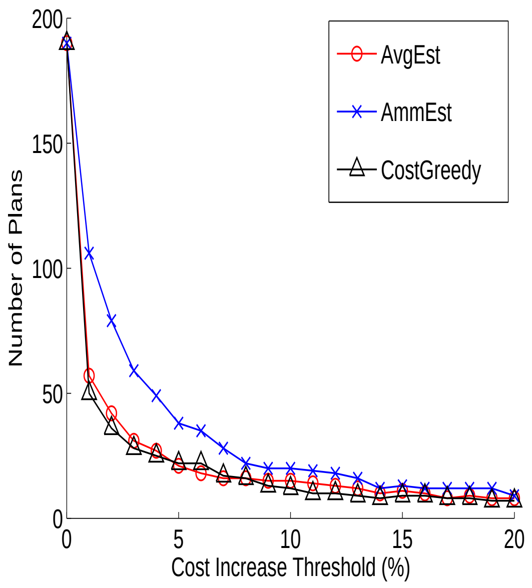
Buffer Pages	Sort Heap	Minimum Cost	Maximum Cost	Original Plans	Reduced Plans (λ = 10%)	Reduced Plans (λ = 20%)
10	10	1.54e4	2.75e7	34	16	14
10	50000	1.54e4	2.71e7	21	10	9
50000	10	1.56e4	6.11e5	37	10	9
50000	50000	1.56e4	5.55e5	27	9	7



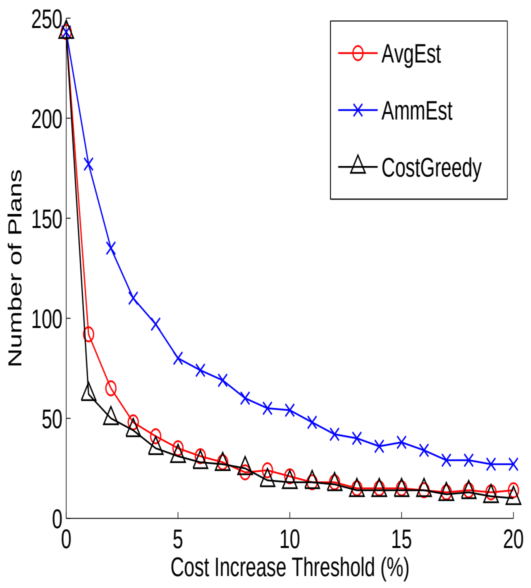
(a) **Est-2D (100)**



(b) **Est-2D (300)**



(c) **Est-3D**



(d) **Est-4D**

Figure 6.6: Estimator Performance

Table 6.9: TPC-DS

TPC-DS Query Template	Original Plans	Reduced Plans (λ = 10%)	Reduced Plans (λ = 20%)
12	13	5	4
17	39	2	2
18	47	11	6
19	36	10	8
25	43	2	2

last two columns of Table 6.8.

For OptC, we found that changing the parameter settings for server memory did not appreciably change the cost of the query points. We intend to investigate this issue further in collaboration with the developers of the OptC database engine.

6.6 TPC-DS

We also validated our results on TPC-DS, the recently released decision support benchmark [46]. TPC-DS models the decision support functions of a retail product supplier, including data loading, multiple types of queries and data maintenance. The database consists of multiple snowflake schemas with shared dimension tables, skewed data and a large query set. We used a 100 GB sample database which has 24 tables, generated using the TPC-DS benchmark's synthetic generator, on OptC. Representative results are shown in Table 6.9 for sample query templates based on the TPC-DS queries. These plan diagrams were produced with 100 query points per dimension, uniformly distributed in the selectivity space. We see in the table that though these plan diagrams are dense, the plan diagram reduction process produces reduced plan diagrams of low cardinality, seemingly *independent* of the properties and complexity of the underlying database.

6.7 Abstract Plan Costing Based Reduction

So far, all the reduced plan diagrams were produced by using the upper-bounding rule specified in Definition 3 (Chapter 3). This made the reduction conservative in terms of the plans that were

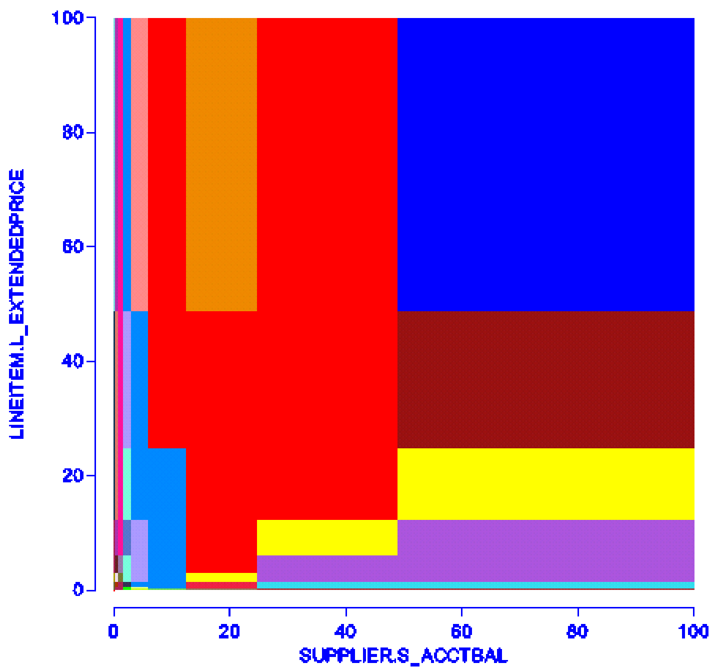
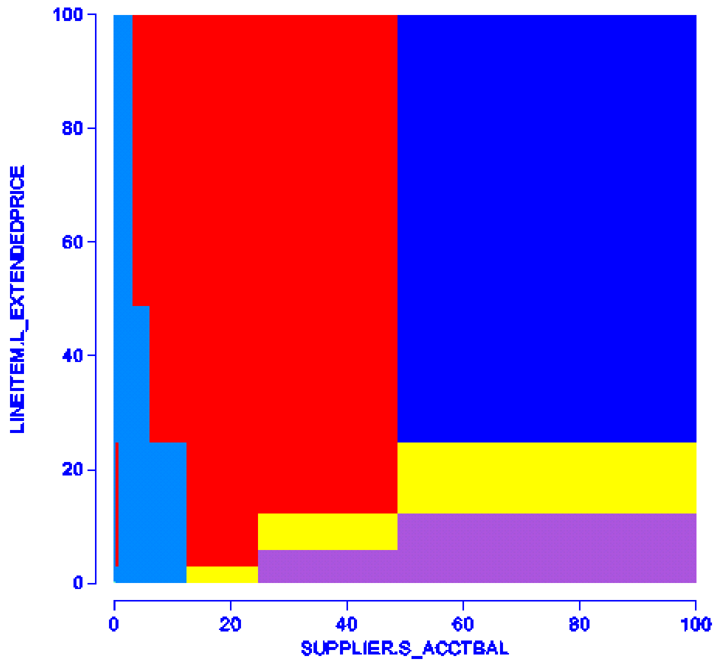


Figure 6.7: Plan Diagram (QT8)

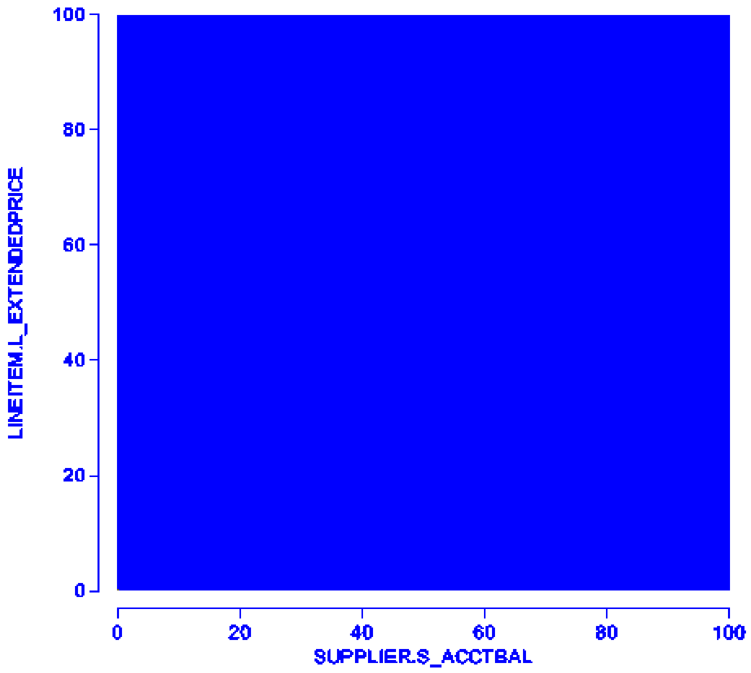
considered as potential replacements for other plans. Some commercial database engines now provide the feature of costing a sub-optimal plan at a query point, which is called the Abstract Plan costing feature. Thus, we can now perform plan diagram reduction without having to adhere to the Cost Bounding Rule. This could enhance reduction as the accurate cost of a potential swallower plan at a query point could comply with the cost-increase threshold in cases where the upper-bound did not. Also, instead of restricting attention to plans in a particular quadrant which can upper bound the cost of replacement, we can find the optimizer-estimated cost of *all* plans in the plan diagram at a query point and choose the best replacement.

The only drawback of this approach is that it is computationally very expensive as the abstract plan of every plan in the original plan diagram has to be extracted and its cost has to be evaluated for every query point in the diagram.

Experimentally, we have observed that Abstract-Plan-based reduction results in a very small number of plans in the reduced plan diagram - sometimes *only one plan*. This one plan is thus close to optimal for all query points, which provides some additional benefits as we will see in Chapter 7. Let us see an example of this reduction. The plan diagram of QT8 with a resolution of 10 and an exponential distribution of query points is shown in Figure 6.7. This diagram has



(a) **Reduced Plan Diagram - CostGreedy**



(b) **Reduced Plan Diagram - Abstract Plan Costing**

Figure 6.8: Reduced Plan Diagram (QT8, $\lambda = 10\%$)

Table 6.10: Computational Efficiency (QT8, Res=100)

Algorithm	**Original Plans**	**Reduced (λ = 10%)**	**Time**
Cost Bounding	50	7	0.17 sec
Abstract Plan based (I quadrant)	50	4	1.25 hrs
Abstract Plan based (All quadrants)	50	3	7.8 hrs

37 plans and cost-bounded reduction (with λ = 10%) results in 5 plans (Figure 6.8(a)), whereas Abstract-Plan-based reduction retains only a single plan as shown in Figure 6.8(b).

Another way to perform Abstract-Plan cost-based reduction is to consider only plans in the first quadrant of a point as potential swallowers. This could help offset some of the computational cost of this method. The times taken by the cost-bounded plan diagram reduction, the abstract-plan-based reduction considering only first-quadrant plans, and the abstract-plan-based reduction considering all plans, are all shown in Table 6.10. It is clear from the table that while abstract-plan-based reduction results in very low cardinality reduced plan diagrams, it is computationally expensive, even when we consider only first quadrant plans as potential swallowers. Specifically, abstract-plan based reduction takes hours in comparison to the seconds taken by cost-bounded reduction. For users willing to bear this increased computational cost, this reduction could provide the ideal solution.

6.8 Rationale for Reduction

Having seen experimental evidence of anorexic plan diagram reduction, we now try to present the intuition behind this phenomenon. It has been observed in [44] that for any query point, the percentage of plans that are within twice the cost of its optimum plan is around 1% of the total number of plans in the search space. Further, our own evaluation of a public-domain optimizer, OptD, has found that the fraction of plans within ten to twenty percent of the optimal is around 0.01 percent. Since the search space is usually in the several tens of thousands, it means that each query point in the plan diagram has a small but definite bucket of close-to-optimal plans associated with it. Further, it is very likely that there are some common plans between neighboring buckets as a plan that is close-to-optimal at a query point usually continues to be so in the neighborhood of the point. Due to this and the fact that the plans within a bucket have nearly equivalent costs, the likelihood of different plans being picked as optimal in a region is high. Thus, a plan that belongs to the optimality space might be only slightly more expensive at some points outside its region of optimality. These are the points that this plan could cover, given a cost increase threshold by the user. This provides the scope for plan diagram reduction.

We have also experimentally observed that the number of close-to-optimal plans is greater for query points that are close to the origin and axes. The presence of more contenders for the optimal plan might explain why the cardinality of plan diagrams is usually greater in these regions.

We intend to investigate the phenomenon of plan diagram reduction more formally in our future work.

Chapter 7

Applications of Plan Diagram Reduction

In the previous chapters, we studied the plan diagram reduction problem and the techniques to perform this reduction. We now turn our attention to sample applications of plan diagram reduction. The numerous benefits that result from reduction of plan diagrams were enumerated in Section 1.4 such as enhancement of PQO usability, identification of least-expected-cost plans, etc. Most of the benefits listed there are proportional to the number of plans eliminated during the plan diagram reduction process, but the identification of error-resistant plans (Section 1.4.3) is a function of the *behavior* of the plans that are retained rather than the number of plans.

7.1 Resistance to errors in selectivity estimates

Plan diagram reduction can help to identify plans that provide robust performance over large regions of the selectivity space. Therefore, *errors* in the underlying database statistics, a situation often encountered by optimizers in practice [25], may have much less impact as compared to using the fine-grained plan choices of the original plan diagram, which may have poor performance at other points in the space.

The estimated location of the query point in the selectivity space could differ from the actual location due to a variety of reasons, such as:

1. Out-of-date statistics: Database engines use statistics to determine the resultant cardinality and thereby the cost of each plan. These statistics, being expensive to maintain, are

usually not continuously kept up-to-date.

2. Coarseness of the statistics: The statistics maintained by the DBMS are inherently coarse. For example, histograms store information in a fixed number of buckets, usually about 20, and the frequency of a value in a bucket is approximated by the average of the frequencies of all values in the bucket.

3. Caching of plans: Some optimizers might cache the plan that is best for the most frequently occurring query point in the selectivity space, and use the plan for all other points.

A selectivity error made by the optimizer can result in one of two situations:

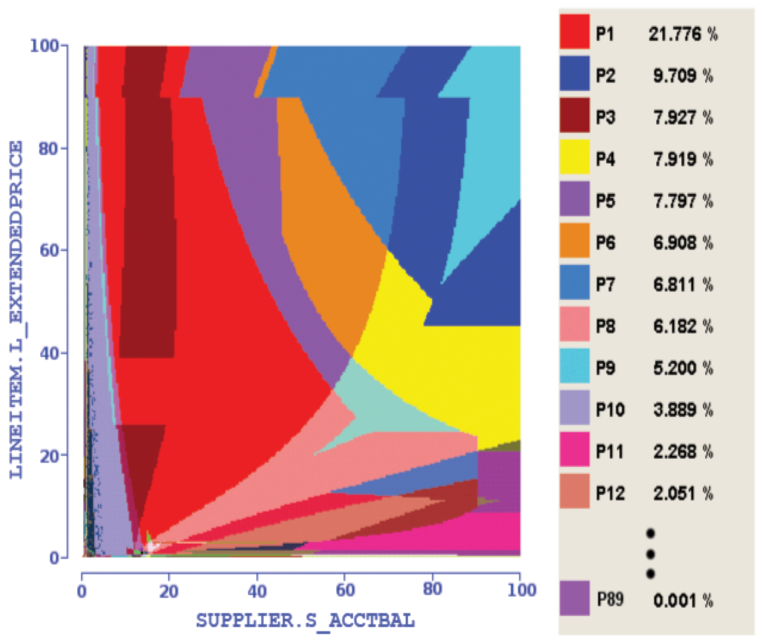
1. The actual location of the point is *within* the optimality region (*endo-optimal*) of the plan that covers the estimated location in the reduced plan diagram.

2. The actual location of the point is *outside* the optimality region (*exo-optimal*) of the plan that covers the estimated location in the reduced plan diagram.

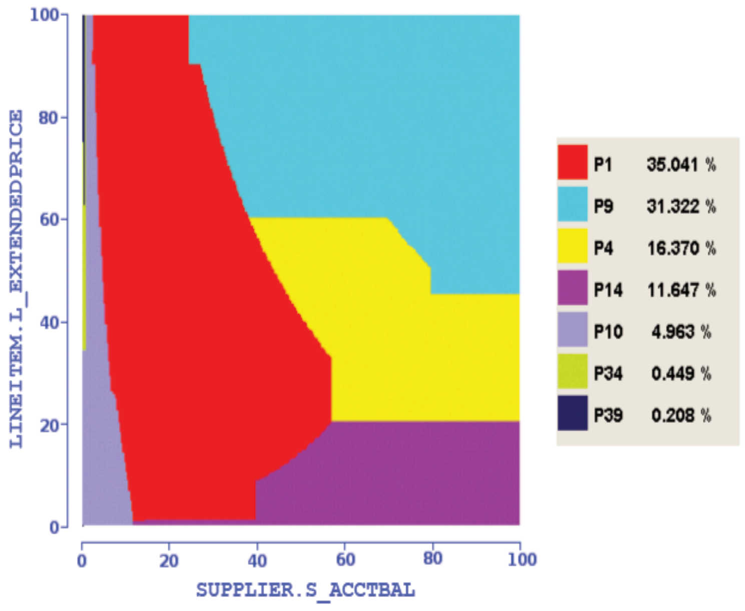
Plan diagram reduction increases the likelihood that a selectivity error will result in the endo-optimal situation. This is due to the fact that each plan that remains in the reduced plan diagram covers an area larger than or equal to the area it covered in the original plan diagram. Reduced plan diagrams are inherently robust to errors of this type as the cost of the replacement plan is guaranteed to be within a λ threshold of the cost of the optimal plan at the actual location.

For example, in Figure 7.1(a) (same as Figure 1.4(a)), estimated selectivities of (14%,1%) leads to a choice of plan P70. However, if the actual selectivities at runtime turn out to be significantly different, say (50%,40%), using plan P70, whose cost increases steeply with selectivity, would be disastrous. In contrast, this error would have had no impact with the reduced plan diagram of Figure 7.1(b) (same as Figure 1.4(d)), since P1, the replacement plan choice at (14%,1%), remains as the preferred plan for a large range of higher values, including (50%,40%). Quantitatively, at (50%, 40%), plan P1 has a cost of 135, while P70 is much more expensive, about *three times* this value.

For selectivity errors that result in the exo-optimal case however, the upper bound on the cost *cannot* be guaranteed. Experimental observation indicates that the plans retained after plan



(a) **Plan Diagram**



(b) **Reduced Diagram ($\lambda = 10\%$)**

Figure 7.1: Sample Plan and Reduced Plan Diagrams (QT8)

diagram reduction tend to be more robust than the ones eliminated by the reduction process. In order to measure this robustness, we define what we call the "*Selectivity-error Resistance Power*" (SeRP) of a replacement plan w.r.t. the original plan at a specific query point.

Consider a query point q with selectivities (x, y), having optimal plan P_i in the original 2-dimensional plan diagram P and assigned plan P_j in the reduced plan diagram R (the extension to n-dimensions is straightforward). Suppose the error in selectivity estimation causes the query point to shift to q' with selectivities (x', y') which was assigned plan P_k in the original plan diagram. Then, SeRP is defined as:

$$SeRP(q, q') = \frac{c_i(q')}{c_k(q')} * \frac{c_i(q') - c_j(q')}{c_i(q') - c_k(q')} \tag{7.1}$$

where $c_i(q)$ is the cost of executing query q with plan P_i.

The first term of the SeRP represents the window of benefit that is available due to the difference in costs between P_i and P_k at q'. The second term measures the fraction of benefit actually garnered by the replacement plan w.r.t. the original plan. The magnitude of the SeRP value denotes the extent of variation between the robustness of the original and replacement plans.

The SeRP is measured across different distances in the selectivity space with the distance representing the error in selectivity estimates. In principle, any distance metric can be used to represent selectivity errors, here we use Euclidean distance. For every distinct pair of points, Figure 7.2 shows the variation of the SeRP for different distances in QT4 on OptC. As we can see here, the reduced plan diagram provides substantially better resistance to errors in selectivity estimates than the original plan diagram, with the SeRP value as high as +23.

7.2 Producing Robust Reduced Plan Diagrams

It should be noted that, in principle, the SeRP can take values from $-\infty$ to $+\infty$. An SeRP value greater than zero indicates that the replacement plan is more robust than the original plan, whereas an SeRP value lesser than zero indicates that the original plan is preferable as compared to the replacement plan. In a plan diagram, the cost of a plan is unknown in its

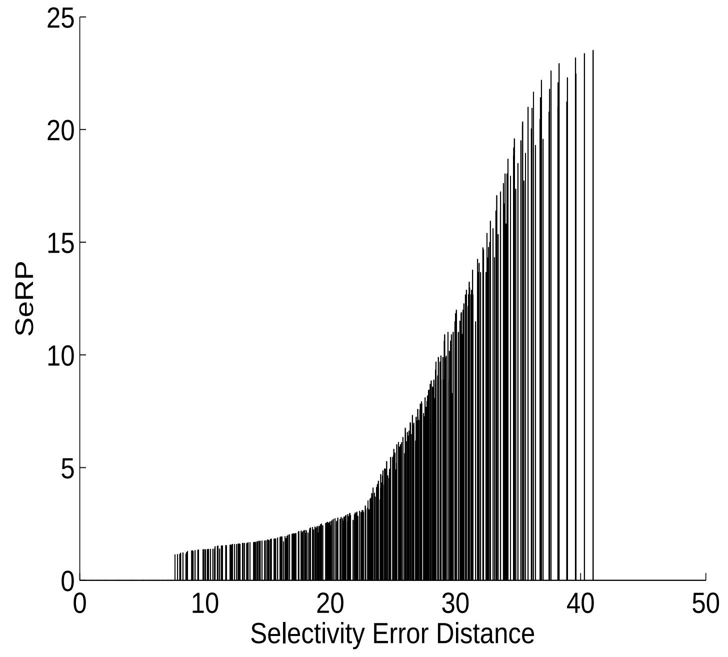


Figure 7.2: SeRP (QT4, $\lambda = 10\%$)

exo-optimal region. Thus, there is no guarantee that the SeRP value for a selectivity error that causes a query point to fall in the exo-optimal region of its replacement plan in the reduced plan diagram will be non-negative. However, if the replacement plan were chosen such that it complies with the λ threshold w.r.t. the original plan not just at the query point of estimated selectivity, but throughout the selectivity space, the SeRP value for this replacement would be bounded. Thus the problem can be stated as follows:

Definition 7 *ErrorResistantPlanRed*

Given an input plan diagram P*, and a cost increase threshold* λ *(*$\lambda \geq 0$*), find a reduced plan diagram* R *that has minimum plan cardinality, and for every plan* P_i *in* P*,*

1. $P_i \in \mathsf{R}$*, or*

2. $\forall$ *query points* $q \in P_i$*,* $\exists P_j \in \mathsf{R}$*, such that* $\forall$ *query points* $q' \in \mathsf{P}$*,*
$$\frac{c_j(q')}{c_i(q')} \leq (1 + \lambda)$$

That is, find the minimum-sized error-resistant "cover" of plans that is sufficient to recolor P (using only the colors in L_P) without increasing the cost of any re-colored query point by

```
RobustCostGreedy (Plan Diagram P, Threshold $\lambda$)

  1. For each point $q$ from $TopRight$ to $BottomLeft$ do
       (a) set $cur(q) = color(q)$
       (b) update $belong(q)$ with plans that are in $q$'s first quadrant with cost within the given threshold
           at q and at all the corners
  2. Let $m = m_1 \times m_2$.
  3. Create $n$ sets $S = \{S_1, S_2, \ldots S_n\}$ corresponding to the $n$ plans.
  4. Let $U = \{1, 2, \ldots m\}$ correspond to the $m$ query points.
  5. Define $\forall i = 1 \ldots n$, $S_i = \{j : i \in belong(r)$ or $i = cur(r)$ for query point $r$ corresponding to $j$,
     $\forall j = 1 \ldots m\}$
  6. Let $I = (U, S)$, $I$ be an instance of the Set Cover problem.
  7. Let $L_n$ be the color of the $TopRight$ point. Remove set $S_n$ and all its elements from $I$.
  8. Apply Algorithm Greedy Setcover to $I$. Let $C$ be the solution found.
  9. $C = C \bigcup \{S_n\}$
 10. Recolor the grid with colors corresponding to the sets in $C$ and update new costs appropriately.
     If a point belongs to more than one subset, then color it with the color that requires the least cost
     increase.
 11. End Algorithm RobustCostGreedy
```

Figure 7.3: Algorithm RobustCostGreedy

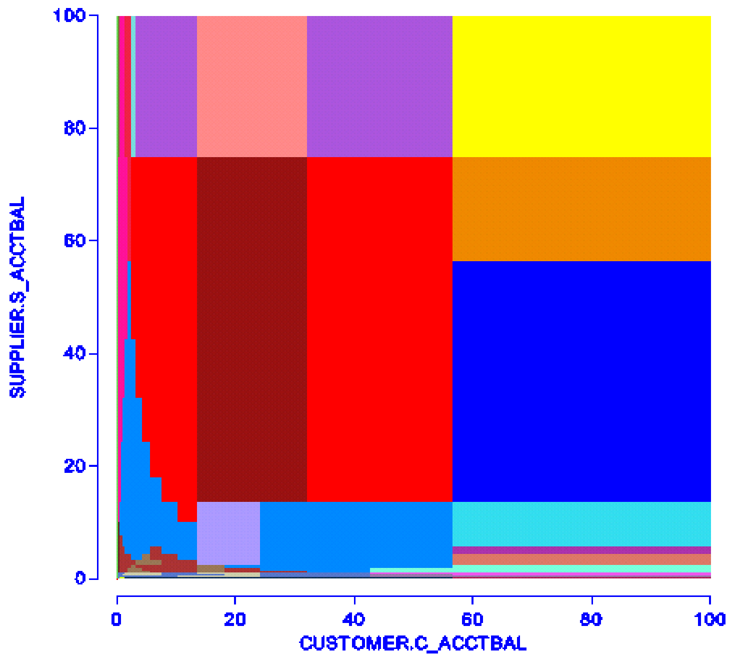


Figure 7.4: Plan Diagram (QT5)

more than the cost increase threshold, *irrespective of the actual location of the query point at run-time*.

In order to find this error-resistant cover of the plan diagram, we need to compare the behavior of each replacement plan with the original plan it is replacing at *all points* in the plan diagram. This requires us to find the cost of this original plan and all potential replacement plans at every point in the diagram. This can be done by using the Abstract-Plan feature explained in Chapter 6. But, as we have already seen there, this feature is extremely expensive. In order to provide a computationally feasible solution, we consider the costs of the original and potential replacement plans at the extreme corner query points of the plan diagram to be representative of the interior points and compare the costs at these points. An extended version of the CostGreedy algorithm called RobustCostGreedy that incorporates this corner-heuristic is shown in Figure 7.3.

For most query templates, our experience has been that the SeRP ≥ 0, indicating that the replacement was beneficial. But, there do exist templates where the replacement proves to be harmful – an extreme case where the SeRP is $\ll 0$ arises with QT5 on OptC. For this combination, the plan diagram (Figure 7.4) contains 51 plans. The reduced plan diagram produced by

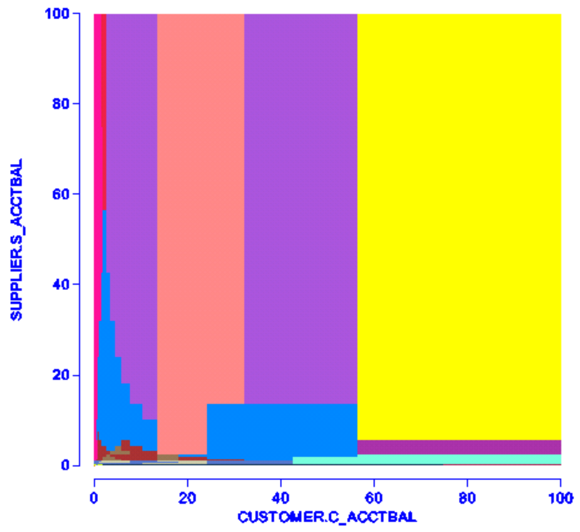
CostGreedy for λ = 10% is shown in Figure 7.5(a), which has 15 plans. The SeRP diagram for this reduction, showing the minimum and maximum SeRP for every pair of points at a given distance is shown in Figure 7.5(b). As is clear from this figure, there are cases where the cost of the replacement plan is orders-of-magnitude greater than that of the original plan at the actual location. In fact, in this case the SeRP takes a value as low as *-2500*!

The average SeRP is computed by summing up all the SeRP values for the corresponding distance and dividing this value by the number of distinct points that occur at this distance. Thus, the average SeRP for a particular distance weighs the benefit to the losses incurred by the replacement across all points that have this selectivity error between them. The average SeRP diagram for this reduction is shown in Figure 7.5(c).

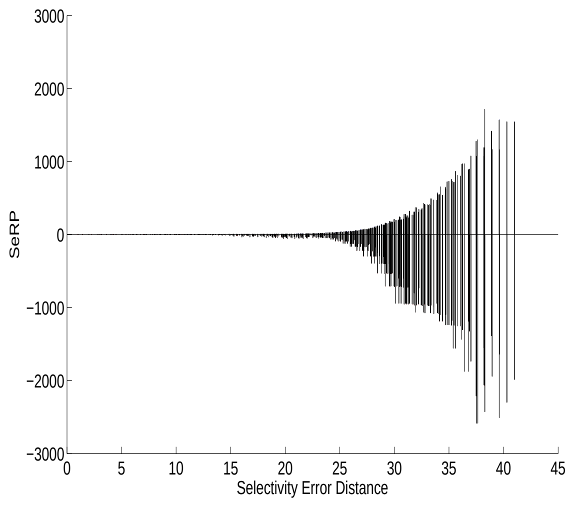
When the reduction is carried out with RobustCostGreedy, the reduced plan diagram shown in Figure 7.6 results. The reduced number of plans in this case is also 15, although some of the plans chosen are different from those chosen by the CostGreedy algorithm. The SeRP diagram for this reduction, showing the minimum and maximum SeRP for every pair of points at a given distance is shown in Figure 7.6(b). The magnitudes of the minimum SeRP values here are significantly greater than those in Figure 7.5(b), which means that the replacement plan does not have a cost much greater than the original plan *irrespective of the error in selectivity estimates*. The average SeRP diagram for this reduction is shown in Figure 7.6(c). The negative SeRP values we saw earlier have almost disappeared here!

Thus, we see that robust plan diagram reduction is feasible and enhances the resistance to selectivity estimate errors in the reduced plan diagram by many orders-of-magnitude. However, this kind of reduction is available only in systems that provide the Abstract-Plan costing feature. Even in these systems, the computational cost of performing this operation prevents us from comparing the cost of the original and replacement plans at all points in the plan diagram.

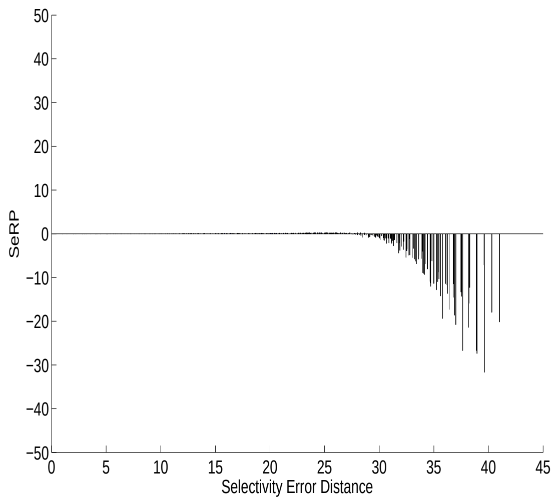
Having demonstrated the benefits provided by abstract-plan-based plan diagram reduction and the enhanced resistance to selectivity estimate errors using abstract-plan costing, we hope that commercial optimizer designers will provide this feature in all optimizers and, further, make it computationally viable to perform many such costings in an efficient manner.



(a) **Reduced Plan Diagram**

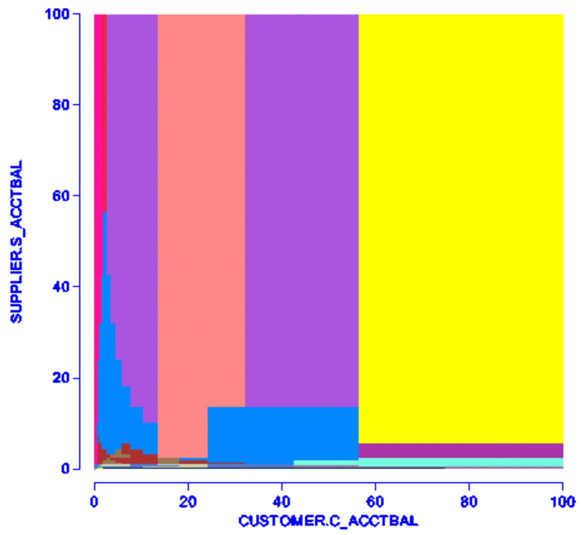


(b) **SeRP Diagram – Min and Max SeRP**

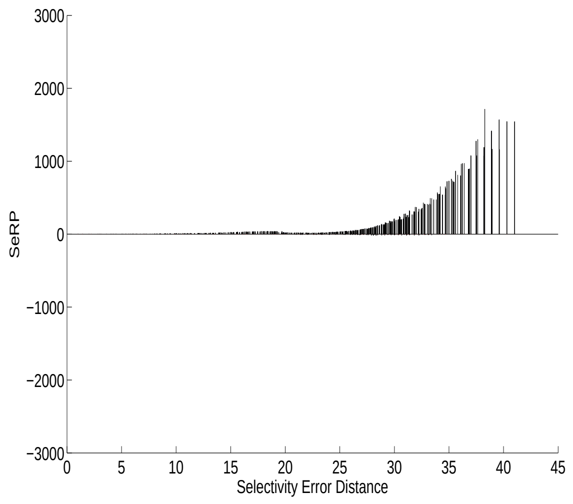


(c) **SeRP Diagram – Average SeRP**

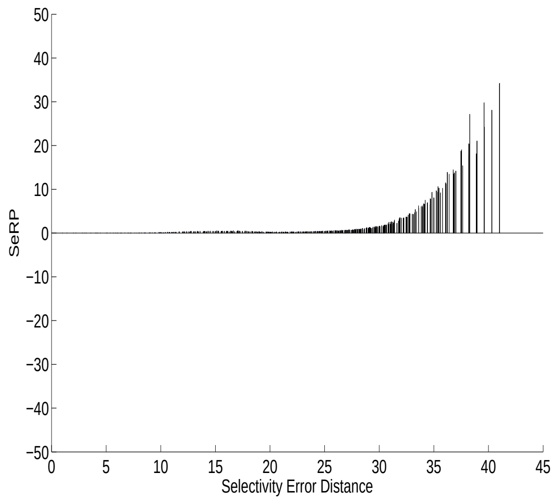
Figure 7.5: Reduced Plan Diagram and SeRP Diagram (QT5, $\lambda = 10\%$)



(a) **Robust Reduced Plan Diagram**



(b) **SeRP Diagram – Min and Max SeRP**



(c) **SeRP Diagram – Average SeRP**

Figure 7.6: Robust Reduced Plan Diagram and SeRP Diagram (QT5, $\lambda = 10\%$)

Chapter 8

Implementation in Picasso

The CostGreedy, ThresholdGreedy, RobustCostGreedy and AmmEst algorithms proposed in this thesis have been implemented in the publicly available query optimizer visualization tool – Picasso v1.0 [30]. Picasso has been developed in the Database Systems Lab [48] at the Indian Institute of Science, for visually analyzing the behavior of industrial-strength relational query optimizers. It generates a host of diagrams that throw light on the functioning of the optimizer for a parameterized query template over the relational selectivity space. Given a query template, the grid resolution, the distribution at which the instances of this template should be spread across the selectivity space, the parameterized relations (axes) and their attributes on which the diagrams should be constructed, and the choice of query optimizer, the Picasso tool automatically generates the associated SQL queries, submits them to the optimizer to generate the plans, and finally produces the color-coded plan, cost and cardinality diagrams.

A block diagram of the Picasso architecture is shown in Figure 8.1. Every request from the user is passed on from the Picasso client to the Picasso server, which handles communication with the database engine and the production of diagrams. The Picasso client is responsible for the visualization of these diagrams. The Picasso server communicates with the database engines through their JDBC interfaces, treating the optimizers as “black boxes”. Picasso currently supports DB2, SQL Server, Oracle, Sybase and PostgreSQL.

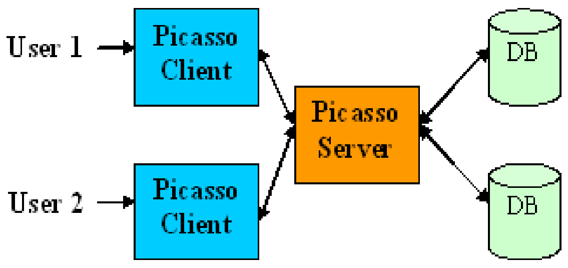


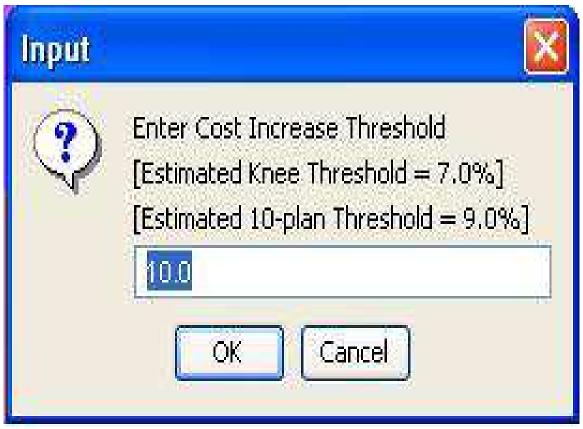
Figure 8.1: Picasso Architecture

8.1 Algorithm Implementation

The CostGreedy algorithm, described in Chapter 4, is implemented in the client module of the Picasso tool. It executes much faster than the AreaGreedy algorithm (Figure 4.1), which was originally used in Picasso v0.5b, as seen in the experimental evaluation of Chapter 6. The CostGreedy algorithm (Figure 4.2) uses the *mincost* data structure explained in Figure 4.4, which is implemented as a 2-dimensional array. The minimum, average and maximum cost increases that result from the plan diagram reduction process are also computed.

The AmmEst algorithm is used to auto-suggest the estimated knee of the plan cardinality vs. cost increase threshold graph and the cost-increase-threshold required to obtain a user-specified number of plans as shown in Figure 8.2. In this example, the user-specified number of plans is 10, which can be changed by the user by setting a parameter in the PicassoConstants file. For each plan, we find the minimum, average and maximum costs of the points associated with it and use these costs as representative of the plan. A plan P_i can be swallowed by a plan P_j if and only if all three cost values of P_j are within threshold of the corresponding cost values of P_i.

The RobustCostGreedy algorithm introduced in Chapter 7 is implemented in the same way as the CostGreedy algorithm. The additional check here is performed by finding the estimated cost of all plans at the four extreme corners of the plan diagram, that is the $TopRight, TopLeft, BottomRight$ and $BottomLeft$ points, and comparing the estimated costs of the original and replacement plans at these points in addition to the point where the



Enter Cost Increase Threshold
[Estimated Knee Threshold = 7.0%]
[Estimated 10-plan Threshold = 9.0%]
10.0

Figure 8.2: Estimator

replacement is being considered. These costs are computed using the Abstract Plan Costing feature described later.

8.2 System Memory Restrictions

We were able to restrict the number of buffer pages and the sort heap size that the optimizer can allocate to the plan in OptA [49]. The command used to do this is "*db2fopt <dbname> update opt_buffpage <value>*".

In OptB, we found the DB_CACHE_SIZE parameter that specifies the minimum buffer size, but could not find one that supports setting the maximum size.

Finally, in OptC, we tried to restrict the memory by altering the parameters "*min server memory*" and "*max server memory*" in the *sys.configurations* table [*personal communication Nicolas Bruno*]. As mentioned earlier, we found that this did not appreciably change the cost of the query points. We intend to investigate this issue further in collaboration with the developers of the OptC database engine.

8.3 Abstract Plan Costing

The Abstract Plan costing based reduction feature explained in Chapter 7 is implemented only on OptC as this feature is not available in the other optimizers. OptC allows the user to specify, as a hint, the plan to be used to execute a given query. It then verifies the validity of the plan for the query and computes the estimated cost of execution. Parameter-level changes might be made to the plan if the optimizer considers this necessary, but generally, the plan is not materially changed.

We use this feature of OptC to find the estimated cost of a plan at a query point where it is sub-optimal. In order to do this, we set the parameter "*showplan_xml*" to *on* and retrieve the plan from the database engine in the form of an XML string. This plan is then appended to the query template as a hint, as shown in Figure 8.3, and queries based on this modified template are now sent to the optimizer to be evaluated. Thus, we obtain the cost of running the query with this plan throughout the selectivity space. The plan diagram is then reduced using these localized costs instead of using the Cost Bounding Rule.

```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)

from (select YEAR(o_orderdate) as o_year, l_extendedprice * (1 - l_discount) as volume,
    n2.n_name as nation

from part, supplier, lineitem, orders, customer, nation n1, nation n2, region

where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey and
    o_custkey = c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey
    and s_nationkey = n2.n_nationkey and r_name = 'AMERICA' and p_type = 'ECONOMY
    ANODIZED STEEL'
    and s_acctbal :varies and l_extendedprice :varies

) as all_nations

group by o_year

order by o_year

–Picasso_Abstract_Plan

option (use plan N'

    <ShowPlanXML xmlns="http://schemas.microsoft.com/sqlserver/2004/07/showplan"
        Version="1.0" Build="9.00.1399.06">
        <BatchSequence>
            <Batch>
                <Statements>
                .
                .
                <OutputList>
                    <ColumnReference Database="[tpch]" Schema="[dbo]" Ta-
                     ble="[LINEITEM]" Column="L_EXTENDEDPRICE"/>
                    <ColumnReference Database="[tpch]" Schema="[dbo]" Ta-
                     ble="[LINEITEM]" Column="L_DISCOUNT"/>
                    <ColumnReference Database="[tpch]" Schema="[dbo]" Ta-
                     ble="[NATION]" Alias="[n1]" Column="N_REGIONKEY"/>
                    <ColumnReference Database="[tpch]" Schema="[dbo]" Ta-
                     ble="[NATION]" Alias="[n2]" Column="N_NAME"/>
                    <ColumnReference Column="Expr1022"/>
                    <ColumnReference Column="Expr1027"/>
                </OutputList>
                .
                .
                </Statements>
            </Batch>
        </BatchSequence>
    </ShowPlanXML>
```

Figure 8.3: Abstract Plan Costing: QT8

Chapter 9

Conclusions

The plan diagrams of industrial-strength database query optimizers are often remarkably complex and dense, with a large number of plans covering the space. In this thesis, we investigated from a variety of perspectives, the problem of reducing the cardinality of dense plan diagrams produced by modern query optimizers, without adversely affecting the query processing quality specified by the user through a cost increase threshold. Plan diagram reduction has useful implications for the design and usage of query optimizers, including quantifying redundancy in the plan search space, enhancing useability of parametric query optimization, identifying error-resistant and least-expected-cost plans, and minimizing the overheads of multi-plan approaches.

Our analysis showed that while finding the optimal reduction was NP-hard, the CostGreedy algorithm proposed here was able to efficiently provide a tight and optimal performance guarantee. Further, the experimental assessment using the data and query templates generated from the TPC-H benchmark on commercial optimizers indicated that, in practice, CostGreedy was always within a plan or two of the optimal, frequently giving the optimal itself. The AvgEst and AmmEst estimators were able to rapidly provide a fairly accurate assessment of the tradeoff between reduced plan cardinality and the cost threshold, helping users to focus on the interesting threshold ranges. Also, the experimental study indicated that the graph of cardinality versus threshold was typically steep and that the number of plans in the reduced plan diagram was likely to be brought down to *anorexic* levels (within/around ten) with cost increase thresholds of just *twenty percent* even for high-dimensional query templates. We also validated these results on the recently released benchmark TPC-DS, which showed that plan diagram reduction is largely independent of the characteristics of the underlying database.

These results are even more striking when we consider that they are *conservative* since a cost bounding rule was used, rather than the optimizer-estimated costs of replacement plans at query points. When the optimizer-estimated costs were used through Abstract-Plan costing, the reduction obtained was much greater – often retaining only a couple of plans over the entire selectivity space.

We also considered a storage-budgeted variant of the plan diagram reduction where the goal was to identify the best set of k plans that would minimize the cost increase in the reduced plan diagram. To solve this problem, we presented ThresholdGreedy, a greedy algorithm that selects plans based on maximizing the benefits obtained by choosing them and saw that Threshold-Greedy is always guaranteed to provide close to *two-thirds of the optimal benefit*.

In some cases, the reduced plan diagram produced by the CostGreedy algorithm could have very poor performance in the presence of errors in selectivity estimates by the optimizer. In order to offset this poor performance, we proposed the RobustCostGreedy algorithm that retains the plans that are resistant to errors in selectivity estimates made by the optimizer, by checking the robustness of the plans at some representative points in the plan diagram. We saw that the reduced plan diagrams produced by this algorithm were inherently more robust than the original plan diagram.

In closing, our study has shown that plan diagram reduction can be carried out efficiently and can bring down the plan cardinality to a manageable number of plans while maintaining acceptable query processing quality. It has also shown that while the optimization process is sensitive to many parameters including query construction, data distribution, memory resources, etc., the reduction process, on the other hand, is relatively indifferent to these factors. We have also demonstrated a sample application of plan diagram reduction that results in improved plan choices by the optimizer.

In summary, this thesis demonstrates that complex plan diagrams can be efficiently converted to anorexic reduced plan diagrams, a result that could have useful implications for the design and use of next-generation database query optimizers.

9.1 Future Work

The work that we have presented in this thesis can be extended in the following ways:

1. In PlanRed, it is required to guarantee that the cost of *every individual* swallowed query point in the original diagram is not increased by more than λ. An alternative problem formulation could be one where it is only required to guarantee that the *average* cost increase in the reduced diagram is not more than λ. The goal here is to come up with heuristic algorithms to efficiently achieve maximum reduction for this alternative metric.

2. A limitation of the RobustCostGreedy algorithm that we have proposed is that, since it only checks the robustness of the replacement plans at the corner points, it is still possible to get a negative SeRP value for some selectivity errors. An alternative approach to provide error-resistance could be to first analyze and characterize the kinds of plan cost functions that arise in industrial-strength optimizers, and then, to investigate the kinds of mathematical cost functions under which it would be possible to provide global guarantees on the cost of a plan in the face of selectivity errors.

3. The PlanRed problem is NP-Hard, as was shown in this thesis. An interesting extension to this problem is to assess whether a fixed-parameter tractable solution [10] can be designed for PlanRed. That is, can a parameter k (which can possibly be the solution size) be identified, leading to an algorithm for optimally solving PlanRed whose running time is polynomial in the input size (number of points and plans in the input plan diagram), while it may be exponential in k.

4. As we have seen in Table 6.10, the Abstract Plan costing based plan diagram reduction, while highly effective, is prohibitively expensive. One of the reasons for this could be that when an abstract plan is sent to the optimizer to be evaluated, the optimizer may also be checking the validity of the plan for the given query. While this might be essential for a plan formulated by the user, if a *plan generated by the optimizer* is returned to it for a different selectivity value, the validity check would be redundant. Therefore, a useful alternative would be to provide the abstract plan costing feature without the validity

check, which could make this method of plan diagram reduction feasible. We hope the results shown here will encourage commercial optimizers to support abstract-plan costing without validity checking.

5. We intend to conduct a more formal investigation of the phenomenon of plan diagram reduction.

6. Finally, and most importantly, to use these results and re-engineer current optimizers to directly produce reduced plan diagrams in the first instance, which could potentially speed up the complex process of query optimization, in addition to providing the other benefits mentioned in this thesis.

Appendix A

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

Figure A.1: QT2

```
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_totalprice :varies
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_extendedprice :varies
    )
group by
    o_orderpriority
order by
    o_orderpriority
```

Figure A.2: QT4

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

Figure A.3: QT5

```
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume)
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            YEAR (l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = ’FRANCE’
                and n2.n_name = ’GERMANY’)
                or (n1.n_name = ’GERMANY’
                and n2.n_name = ’FRANCE’)
            )
            and l_shipdate between ’1995-01-01’
             and ’1996-12-31’
            and o_totalprice :varies
            and c_acctbal :varies
    )as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year
```

Figure A.4: QT7

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

Figure A.5: QT8

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

Figure A.6: QT9

```
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= ’1993-10-01’
    and o_orderdate < ’1994-01-01’
    and c_nationkey = n_nationkey
    and c_acctbal :varies
    and l_extendedprice :varies
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
```

Figure A.7: QT10

```
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_retailprice :varies
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
            and l_extendedprice :varies
    )
```

Figure A.8: QT17

```
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
from
    customer,
    orders,
    lineitem
where
    o_orderkey in (
        select
            l_orderkey
        from
            lineitem
        where l_extendedprice :varies
        group by
            l_orderkey having
                sum(l_quantity) > 300
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
    and c_acctbal :varies
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate
```

Figure A.9: QT18

```
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and exists (
        select
            *
        from
            lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
        select
            *
        from
            lineitem l3
        where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey <> l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and s_acctbal :varies
    and l1.l_extendedprice :varies
    and n_name = 'SAUDI ARABIA'
group by
    s_name
order by
    numwait desc,
    s_name
```

Figure A.10: QT21

```
select
    i_item_desc,
    i_category,
    i_class,
    i_current_price,
    sum(ws_ext_sales_price) as itemrevenue,
    sum(ws_ext_sales_price)*100/sum(sum(ws_ext_sales_price))
    over (partition by i_class) as revenueratio
from
    web_sales,
    item,
    date_dim
where
    ws_item_sk = i_item_sk
    and ws_sold_date_sk = d_date_sk
    and d_date between ’1998-05-16’ and ’998-06-16’
    and i_current_price :varies
    and ws_list_price :varies
group by
    i_item_id,
    i_item_desc,
    i_category,
    i_class,
    i_current_price
order by
    i_category,
    i_class,
    i_item_id,
    i_item_desc,
    revenueratio
```

Figure A.11: DSQT12

```
select
    i_item_id,
    i_item_desc,
    s_state,
    count(ss_quantity) as store_sales_quantitycount,
    avg(ss_quantity) as store_sales_quantityave,
    stdev(ss_quantity) as store_sales_quantitystdev,
    stdev(ss_quantity)/avg(ss_quantity)
     as store_sales_quantitycov,
    count(sr_return_quantity)
     as_store_returns_quantitycount,
    avg(sr_return_quantity)
     as_store_returns_quantityave,
    stdev(sr_return_quantity)
     as_store_returns_quantitystdev,
    stdev(sr_return_quantity)/avg(sr_return_quantity)
     as store_returns_quantitycov,
    count(cs_quantity) as catalog_sales_quantitycount,
    avg(cs_quantity) as catalog_sales_quantityave,
    stdev(cs_quantity)/avg(cs_quantity)
     as catalog_sales_quantitystdev,
    stdev(cs_quantity)/avg(cs_quantity)
     as catalog_sales_quantitycov
from
    store_sales, store_returns, catalog_sales,
    date_dim d1, date_dim d2, date_dim d3, store, item
where
    d1.d_quarter_name = ’2002Q1’
    and d1.d_date_sk = ss_sold_date_sk
    and i_item_sk = ss_item_sk
    and s_store_sk = ss_store_sk
    and ss_customer_sk = sr_customer_sk
    and ss_item_sk = sr_item_sk
    and ss_ticket_number = sr_ticket_number
    and sr_returned_date_sk = d2.d_date_sk
    and d2.d_quarter_name in (’2002Q1’,’2002Q2’,’2002Q3’)
    and sr_customer_sk = cs_bill_customer_sk
    and sr_item_sk = cs_item_sk
    and cs_sold_date_sk = d3.d_date_sk
    and d3.d_quarter_name in (’2002Q1’,’2002Q2’,’2002Q3’)
    and ss_list_price :varies
    and cs_list_price :varies
group by
    i_item_id, i_item_desc, s_state
order by
    i_item_id, i_item_desc, s_state
```

Figure A.12: DSQT17

```
select
    i_item_id,
    ca_country,
    ca_state,
    ca_county,
    avg(cs_quantity) agg1,
    avg(cs_list_price) agg2,
    avg(cs_coupon_amt) agg3,
    avg(cs_sales_price) agg4,
    avg(cs_net_profit) agg5,
    avg(c_birth_year) agg6,
avg(cd1.cd_dep_count) agg7
from
    catalog_sales,
    customer_demographics cd1,
    customer_demographics cd2,
    customer,
    customer_address,
    date_dim,
    item
where
    cs_sold_date_sk = d_date_sk
    and cs_item_sk = i_item_sk
    and cs_bill_cdemo_sk = cd1.cd_demo_sk
    and cs_bill_customer_sk = c_customer_sk
    and cd1.cd_gender = ’F’
    and cd1.cd_education_status = ’Unknown’
    and c_current_cdemo_sk = cd2.cd_demo_sk
    and c_current_addr_sk = ca_address_sk
    and c_birth_month in (3,11,9,5,8,10)
    and d_year = 2000
    and ca_state in (’NC’,’AK’,’PA’,’AK’,’CA’,’MA’,’WV’)
    and cs_list_price :varies
    and i_current_price :varies
group by
    i_item_id,
    ca_country,
    ca_state,
    ca_county
order by
    ca_country,
    ca_state,
    ca_county
```

Figure A.13: DSQT18

```
select
    i_brand_id brand_id,
    i_brand brand,
    i_manufact_id,
    i_manufact,
    sum(ss_ext_sales_price) ext_price
from
    date_dim,
    store_sales,
    item,
    customer,
    customer_address,
    store
where
    d_date_sk = ss_sold_date_sk
    and ss_item_sk = i_item_sk
    and d_moy=12
    and d_year=1999
    and ss_customer_sk = c_customer_sk
    and c_current_addr_sk = ca_address_sk
    and substring(ca_zip,1,5) <> substring(s_zip,1,5)
    and ss_store_sk = s_store_sk
    and ss_list_price :varies
    and i_current_price :varies
group by
    i_brand,
    i_brand_id,
    i_manufact_id,
    i_manufact
order by
    ext_price desc,
    i_brand,
    i_brand_id,
    i_manufact_id,
    i_manufact
```

Figure A.14: DSQT19

```
select
    i_item_id,
    i_item_desc,
    s_store_id,
    s_store_name,
    sum(ss_net_profit) as store_sales_profit,
    sum(sr_net_loss) as store_returns_loss,
    sum(cs_net_profit) as catalog_sales_profit
from
    store_sales,
    store_returns,
    catalog_sales,
    date_dim d1,
    date_dim d2,
    date_dim d3,
    store,
    item
where
    d1.d_moy = 4
    and d1.d_year = 1999
    and d1.d_date_sk = ss_sold_date_sk
    and i_item_sk = ss_item_sk
    and s_store_sk = ss_store_sk
    and ss_customer_sk = sr_customer_sk
    and ss_item_sk = sr_item_sk
    and ss_ticket_number = sr_ticket_number
    and sr_returned_date_sk = d2.d_date_sk
    and d2.d_moy between 4 and  4+6
    and d2.d_year = 1999
    and sr_customer_sk = cs_bill_customer_sk
    and sr_item_sk = cs_item_sk
    and cs_sold_date_sk = d3.d_date_sk
    and d3.d_moy between 4 and 4+6
    and d3.d_year = 1999
    and ss_list_price :varies
    and cs_list_price :varies
group by
    i_item_id,
    i_item_desc,
    s_store_id,
    s_store_name
order by
    i_item_id,
    i_item_desc,
    s_store_id,
    s_store_name;
```

Figure A.15: DSQT25

References

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

[2] B. Babcock and S. Chaudhuri, “Towards a Robust Query Optimizer: A Principle and Practical Approach”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2005.

[3] S. Babu, P. Bizarro, D. DeWitt, “Proactive Re-Optimization”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2005.

[4] A. Betawadkar, “Query Optimization with One Parameter”, *Master’s Thesis, Dept. of Computer Science & Engineering, IIT Kanpur*, February 1999.

[5] S. Chaudhuri, “An Overview of Query Optimization in Relational Systems”, *Proc. of ACM Principles of Database Systems (PODS)*, June 1998.

[6] F. Chu, J. Halpern and P. Seshadri, “Least Expected Cost Query Optimization: An Exercise in Utility”, *Proc. of ACM Principles of Database Systems (PODS)*, May 1999.

[7] F. Chu, J. Halpern and J. Gehrke, “Least Expected Cost Query Optimization: What Can We Expect”, *Proc. of ACM Principles of Database Systems (PODS)*, May 2002.

[8] R. Cole and G. Graefe, “Optimization of Dynamic Query Evaluation Plans”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1994.

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

[10] R. Downey, M. Fellows and U. Stege, “Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability”, *DIMACS Vol. 49*, 1999.

[11] U. Feige, “A threshold of ln n for approximating set cover”, *Journal of ACM*, 45(4), 1998.

[12] S. Ganguly, “Design and Analysis of Parametric Query Optimization Algorithms”, *Proc. of 24th Intl. Conf. on Very Large Data Bases (VLDB)*, August 1998.

[13] 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.

[14] M. Garey and D. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W. H. Freeman & Co, 1979.

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

[16] 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.

[17] G. Graefe and D. DeWitt, “The Exodus Optimizer Generator”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 1987.

[18] G. Graefe, “Query Evaluation Techniques for Large Databases”, *ACM Computing Survey*, 1993.

[19] A. Hulgeri and S. Sudarshan, “Parametric Query Optimization for Linear and Piecewise Linear Cost Functions”, *Proc. of 28th Intl. Conf. on Very Large Data Bases (VLDB),* August 2002.

[20] 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 (VLDB),* September 2003.

[21] I. Ilyas et al, “Estimating Compilation Time of a Query Optimizer”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2003.

[22] Y. E. Ioannidis and Y. C. Kang, “Left-Deep vs. Bushy Trees: An Analysis of Strategy Spaces and its Implications for Query Optimization”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1991.

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

[24] M. Jarke and J. Koch. “Query optimization in database systems” ACM Computing Surveys, 16(2):111-152, June 1984.

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

[26] D. Kossmann and K. Stocker, “Iterative dynamic programming: a new class of query optimization algorithms”, *ACM Trans. on Database Systems (TODS),* December 2000.

[27] L. Mackert and G. Lohman, “R* Optimizer Validation and Performance Evaluation for Local Queries”, *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1986.

[28] V. Markl et al, “Robust Query Processing through Progressive Optimization” *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2004.

[29] J. Melton and A. Simon, “Understanding The New SQL: A Complete Guide”, *Morgan Kaufmann*, May 1993.

[30] Picasso Database Query Optimizer Visualizer, *http://dsl.serc.iisc.ernet.in/projects/PICASSO/inde*

[31] T. Ramsinghani, “Picasso 1.0: Design and Analysis”, *Master’s Thesis, Dept. of Computer Science and Automation, IISc Bangalore*, July 2007.

[32] S. Rao, “Parametric Query Optimization: A Non-Geometric Approach”, *Master’s Thesis, Dept. of Computer Science & Engineering, IIT Kanpur*, March 1999.

[33] N. Reddy and J. Haritsa, “Analyzing Plan Diagrams of Database Query Optimizers”, *Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB),* August 2005.

[34] N. Reddy, “Next Generation Relational Query Optimizers”, *Master’s Thesis, Dept. of Computer Science and Automation, IISc Bangalore*, June 2005.

[35] 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.

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

[37] 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 (VLDB)*, September 2004.

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

[39] 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.

[40] E. Shekita and H. Young, “Iterative Dynamic Programming”, *IBM Tech. Report*, 1998.

[41] A. Silberschatz, H. Korth and S. Sudarshan, “Database System Concepts”, McGrawHill, 1997

[42] P. Slavik, “A tight analysis of the greedy algorithm for set cover”, *Proc. of 28th ACM Symp. on Theory of Computing*, 1996.

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

[44] 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.

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

[46] *http://www.tpc.org/tpcds*

[47] *http://en.wikipedia.org/wiki/Bellman_equation*

[48] *http://dsl.serc.iisc.ernet.in/*

[49] *http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0508kapoor/*

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