SIGHT and SEER: Efficient Production and Reduction of Query Optimizer Plan Diagrams

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

by

Harish D

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

July 2008

To my Parents, whose dedication to my success and their continued support,

I shall always remember.

Acknowledgements

I would like to thank my advisor Prof. Jayant Haritsa for introducing me to this problem and providing valuable guidance and encouragement. I would like to sincerely acknowledge his invaluable support in all forms through out my stay in IISc.

I would like to thank all the members of DSL and CSA who have made my stay at IISc memorable. I thank my family and friends for their continued support throughout my career.

Publications

1. Harish D., Pooja N. Darera, Jayant R. Haritsa
   "Identifying Robust Plans through Plan Diagram Reduction"
   Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB), Auckland, New Zealand, August 2008.

2. Atreyee Dey, Sourjya Bhaumik, Harish D., Jayant R. Haritsa
   "Efficiently Approximating Query Optimizer Plan Diagrams"
   Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB), Auckland, New Zealand, August 2008.

3. Harish D., Pooja N. Darera, Jayant R. Haritsa
   "Robust Plans through Plan Diagram Reduction", Technical Report, TR-2007-02, DSL/SERC, Indian Institute of Science.
   `http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2007-02.pdf`

4. Atreyee Dey, Sourjya Bhaumik, Harish D., Jayant R. Haritsa
   "Efficient Generation of Approximate Plan Diagrams", Technical Report, TR-2008-01, DSL/SERC, Indian Institute of Science.
   `http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2008-01.pdf`

Abstract

Given a parametrized n-dimensional SQL query template and a choice of query optimizer, a plan diagram is a color-coded pictorial enumeration of the execution plan choices of the optimizer over the query parameter space. These diagrams have proved to be a powerful metaphor for the analysis and redesign of modern optimizers, and are gaining currency in diverse industrial and academic institutions. However, their utility is adversely impacted by the impractically large computational overheads incurred when standard brute-force exhaustive approaches are used for producing high-dimension and high-resolution diagrams.

In this thesis, we investigate strategies for efficiently producing high-quality approximate plan diagrams that have low plan-identity and plan-location errors. Through experimentation with a representative set of TPC-H-based query templates on a customized public domain optimizer, we show that our techniques are capable of meeting identity and location error bounds as low as 10% while incurring less than 5% of the computational overheads of the exhaustive approach. In fact, we can virtually guarantee zero error with overheads of less than 10%.

The second problem we address in this thesis is that of identifying robust plans using plan diagram reduction. Several plans in these plan diagrams frequently result in inflated query response times due to errors in predicate selectivities estimated by the database query optimizer, which often differ significantly from those actually encountered during query execution. We investigate here mitigating this problem by replacing selectivity error-sensitive plan choices with alternative plans that provide robust performance. Our approach is based on the recent observation that even the complex and dense plan diagrams associated with industrial-strength optimizers can be efficiently reduced to “anorexic” equivalents featuring only a few plans,

without materially impacting query processing quality.

Extensive experimentation with a representative set of TPC-H and TPC-DS-based query templates on a commercial optimizer indicates that plan diagram reduction typically retains plans that are substantially resistant to selectivity errors on the base relations. However, it can sometimes also be severely counter-productive, with the replacements performing much worse. We address this problem through a generalized mathematical characterization of plan cost behavior over the parameter space, which lends itself to efficient criteria of when it is safe to reduce. Our strategies are fully non-invasive and have been implemented in the Picasso optimizer visualization tool.

In summary, we present in this thesis efficient plan diagram generation techniques, followed by effective strategies to substantially increase resistance to selectivity errors by identifying robust plans through plan diagram reduction.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modern database systems use a query optimizer to identify the most efficient strategy to execute the SQL queries that are submitted by users. The efficiency of the strategies, called "plans", is usually measured in terms of query response time. Optimization is a mandatory exercise since the difference between the cost of the best execution plan and a random choice could be in orders of magnitude. The role of query optimizers has become especially critical in recent times due to the high degree of query complexity characterizing current data warehousing and mining applications, as exemplified by the TPC-H [26] and the recent TPC-DS [27] benchmarks.

While commercial query optimizers each have their own "secret sauce" to identify the best plan, the de-facto standard underlying strategy, based on the classical System R optimizer [22], is the following: Given a user query, apply a variety of heuristics to restrict the combinatorially large search space of plan alternatives to a manageable size; estimate, with a cost model and a dynamic-programming based algorithm, the efficiency of each of these candidate plans; finally, choose the plan with the lowest estimated cost.

1.1 Plan Diagrams and Reduced Plan Diagrams

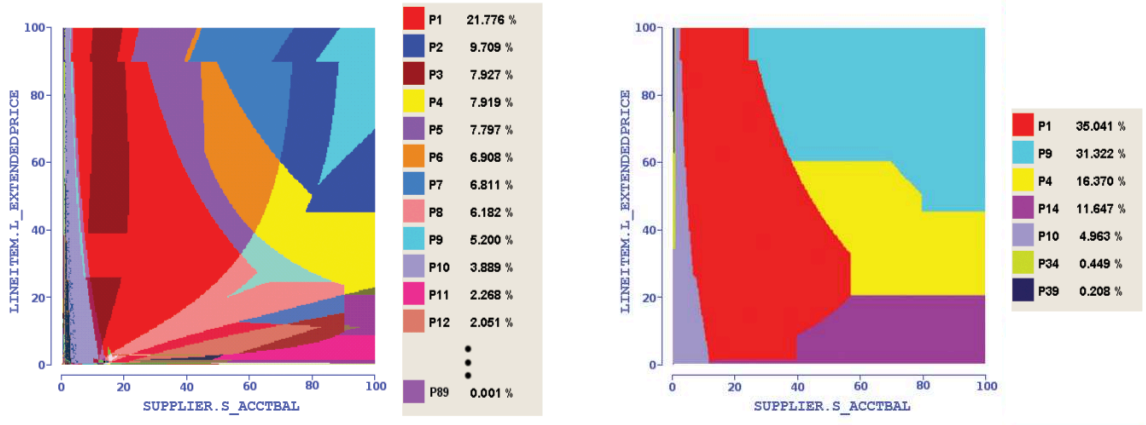
For a given database and system configuration, a query optimizer's execution plan choices are primarily a function of the *selectivities* of the base relations in the query. A "plan diagram" is

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

Figure 1.1: Example Query Template: QT8

a color-coded pictorial enumeration of the plan choices of the optimizer for a parameterized query template over the relational selectivity space. For example, consider QT8, the parameterized 2D query template shown in Figure 1.1, based on Query 8 of TPC-H. Here, selectivity variations on the SUPPLIER and LINEITEM relations are specified through the s_acctbal :varies and l_ extendedprice :varies predicates, respectively. The associated plan diagram for QT8 is shown in Figure 1.2(a), produced with the Picasso optimizer visualization tool [20] on a popular commercial database engine.

In this picture, each colored region represents a specific plan, and a set of 89 different optimal plans, P1 through P89, cover the selectivity space. The value associated with each plan in the legend indicates the percentage area covered by that plan in the diagram – the biggest, P1, for example, covers about 22% of the space, whereas the smallest, P89, is chosen in only 0.001% of the space. ***[Note to Readers: We recommend viewing all diagrams presented in this thesis directly from the color PDF file, or from a color print copy, since the greyscale version may not clearly register the various features.]***



(a) **Plan Diagram** (b) **Reduced Diagram (Threshold = 10%)**

Figure 1.2: Sample Plan Diagram and Reduced Plan Diagram (QT8)

As evident from Figure 1.2(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 benchmark-based query templates on industrial-strength optimizers are available at [20]. However, these dense diagrams can typically be "reduced" to much simpler pictures featuring significantly fewer plans, *without materially degrading the processing quality of any individual query*. For example in Figure 1.2(a), if users are willing to tolerate a minor cost increase (λ) of at most 10% for any query point in the diagram, relative to its original cost, the picture could be reduced to Figure 1.2(b), 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.

A detailed study of the plan diagram reduction problem was presented in [11], and it was shown that a cost increase threshold of *only 20 percent* is usually amply sufficient to bring down the absolute number of plans in the final reduced picture to *within or around ten*. In short, that complex plan diagrams can be made "anorexic" while retaining acceptable query processing performance.

In this thesis, we solve the following two problems: (1) Efficiently generating plan diagrams, and (2) Identifying robust plans using plan diagram reduction.

1.2 Problem I: Efficient Generation of Plan Diagrams

Since their introduction in 2005 [21], plan diagrams have proved to be a powerful metaphor for the analysis and redesign of industrial-strength database query optimizers. Through our interactions with industrial developers, we have found that these complex diagrams have proved to be quite contrary to the prevailing conventional wisdom. While developers had certainly been extensively analyzing optimizer behavior on *individual queries*, plan diagrams provide a completely different perspective of behavior *over an entire space*, vividly capturing plan transition boundaries and optimality geometries. So, in a literal sense, they deliver the "big picture".

Plan diagrams are currently being used in various industrial and academic sites for a diverse set of applications including analysis of existing optimizer designs; visually carrying out optimizer regression testing; debugging new query processing features; comparing the behavior between successive optimizer versions; investigating the structural differences between neighboring plans in the space; investigating the variations in the plan choices made by competing optimizers; etc. Visual examples of non-monotonic cost behavior in commercial optimizers, indicative of modeling errors, were highlighted in [21].

A particularly compelling immediate utility of plan diagrams is that they provide the input to "plan diagram reduction" algorithms. Anorexic plan diagram reduction has significant practical benefits [11], including quantifying the redundancy in the plan search space, enhancing the applicability of parametric query optimization (PQO) techniques [12, 13], identifying error-resistant and least-expected-cost plans [7, 8], and minimizing the overhead of multi-plan approaches [2, 15].

1.2.1 Generating Plan Diagrams

The generation and analysis of plan diagrams has been facilitated by our development of the Picasso optimizer visualization tool [20]. Given a multi-dimensional SQL query template like QT8 and a choice of database engine, the Picasso tool automatically produces the associated plan diagram. It is operational on several major platforms including IBM DB2, Oracle, Microsoft SQL Server, Sybase ASE and PostgreSQL. The tool, which is freely downloadable, is

now in use by the development groups of several major database vendors, as also by leading industrial and academic research labs worldwide.

The diagram production strategy used in Picasso is the following: Given a d-dimensional query template and a plot resolution of r, the Picasso tool generates r^d queries that are either uniformly or exponentially (user's choice) distributed over the selectivity space. Then, for each of these query points, based on the associated selectivity values, a query with the appropriate constants instantiated is submitted to the query optimizer to be "explained" – that is, to have its optimal plan computed. After the plans corresponding to all the points are obtained, a different color is associated with each unique plan, and all query points are colored with their associated plan colors. Then, the rest of the diagram is colored by painting the region around each point with the color corresponding to its plan. For example, in a 2D plan diagram with a uniform grid resolution of 10, there are 100 real query points, and around each such point a square of dimension 10x10 is painted with the point's associated plan color.

The above exhaustive approach is eminently acceptable for diagrams with few dimensions (upto 2D) and low resolutions (upto 100). However, it becomes impractically expensive for higher dimensions and resolutions due to the exponential growth in overheads. For example, a 3D plan diagram with a resolution of 100 on each selectivity dimension, requires invoking the optimizer 100^3 times – that is, a *million* optimizations have to be carried out. Even with a conservative estimate of about half-second per optimization, the total time required to produce the picture is close to a week! Therefore, although plan diagrams have proved to be extremely useful, their high-dimensional and high-resolution versions pose serious computational challenges.

1.2.2 The SIGHT Algorithm

In this thesis, we address this issue using SIGHT (Selective Incremental Generation of plan diagrams using HisTory), an algorithm to efficiently produce *accurate* plan diagrams. We also investigate the possibility of producing *high-quality approximations* to plan diagrams requiring extremely low overheads in the ISIGHT algorithm, an inexact variant of the SIGHT algorithm. Denoting the true plan diagram as P and the approximation as A, there are two categories of

errors that arise in this process:

Plan Identity Error (ϵ_I): This error metric refers to the possibility of the approximation missing out on a subset of the plans present in the true plan diagram. It is computed as the percentage of plans lost in A relative to P.

The ϵ_I error is challenging to control since a majority of the plans in the plan diagrams, as seen in Figure 1.2(a), are very small in area, and therefore hard to find.

Plan Location Error (ϵ_L): This error metric refers to the possibility of incorrectly assigning plans to query points in the approximate plan diagram. It is computed as the percentage of incorrectly (relative to P) assigned points in A.

The ϵ_L error is also challenging to control since the plan boundaries, as seen in Figure 1.2(a), can be highly non-linear, and are sometimes even irregular in shape [20].

In the remainder of this thesis, we will use ϵ to denote both ϵ_I and ϵ_L. Accurate plan diagrams will have $\epsilon = 0$.

1.2.3 Results

The SIGHT algorithm generates accurate plan diagrams with around *10% overheads* of the brute-force exhaustive method. We improve this in the ISIGHT algorithm, where approximate plan diagrams having values of ϵ around 10% can be generated incurring overheads *less than 5%*.

1.3 Problem II: Identifying Robust Plans through Plan Diagram Reduction

The query execution plan choices made by database engines often turn out to be poor in practice because the optimizer's selectivity estimates are significantly in error with respect to the actual values encountered during query execution. Such errors, which can even be in orders of

magnitude in real database environments [18], arise due to a variety of reasons [24], including outdated statistics, attribute-value independence assumptions and coarse summaries.

1.3.1 Robust Plans

To address this problem, one obvious approach is to improve the quality of the statistical meta-data, for which several techniques have been presented in the literature ranging from improved summary structures [1] to feedback-based adjustments [24] to on-the-fly reoptimization of queries [15, 18, 4]. A complementary and conceptually different approach, which we consider in this thesis, is to identify *robust plans* that are relatively less sensitive to such selectivity errors. In a nutshell, to "aim for resistance, rather than cure", by identifying plans that provide comparatively good performance over large regions of the selectivity space. Such plan choices are especially important for industrial workloads where global stability is as much a concern as local optimality [17].

Related Work

Over the last decade, a variety of *compile-time* strategies have been proposed to identify robust plans. For example, in the Least Expected Cost (LEC) approach [7, 8], it is assumed that the distribution of predicate selectivities is apriori available, and then the plan that has the least-expected-cost over the distribution is chosen for execution. While the performance of this approach is likely to be good on average, it could be arbitrarily poor for a specific query as compared to the optimizer's optimal choice for that query. Moreover, it may not always be feasible to provide the selectivity distributions.

An alternative Robust Cardinality Estimation (RCE) strategy proposed in [3] is to model the selectivity dependency of the cost functions of the various competing plan choices. Then, given a user-specified "confidence threshold" T, the plan that is expected to have the *least upper bound* with regard to cost in T percentile of the queries is selected as the preferred choice. The choice of T determines the level of risk that the user is willing to sustain with regard to worst-case behavior. Like the LEC approach, this too may be arbitrarily poor for a specific query as compared to the optimizer's optimal choice.

Finally, in the (initial) optimization phase of the Rio approach [4, 5], a set of uncertainty modeling rules from [15] are used to classify selectivity errors into one of six categories (ranging from "no uncertainty" to "very high uncertainty") based on their derivation mechanisms. Then, these error categories are converted to hyper-rectangular error boxes drawn around the optimizer's point estimate. Finally, if the plans chosen by the optimizer at the corners of the principal diagonal of the box are the same as that chosen at the point estimate, then this plan is *assumed* to be robust throughout the box. However, the conditions under which this assumption is likely to be valid are not outlined.

The above techniques have provided novel and elegant formulations, but have to contend with the following issues:

1. They are *intrusive* requiring, to varying degrees, modifications to the optimizer engine.
2. They require *specialized* information about the workload and/or the system which may not always be easy to obtain or model.
3. Their query capabilities may be *limited* compared to the original optimizer – e.g., only SPJ queries with key-based joins were considered in [3, 4]. Further, [4] has been implemented and evaluated on a non-commercial optimizer.
4. Most importantly, as explained in Section 1.3.1, none of them provide, on an individual query basis, quantitative *guarantees* on the quality of their final plan choice relative to the original (unmodified) optimizer's selection. That is, they "cater to the crowd, not individuals".

1.3.2 The SEER Algorithm.

In this thesis, we present **SEER** (Selectivity-Estimate-Error-Resistance), a new strategy for identifying robust plans that can be directly used on operational database environments. More concretely, it

- Treats the optimizer as a black-box and therefore is inherently (a) completely non-intrusive, and (b) capable of handling whatever SQL is supported by the system. Further,

it does not expect to have any additional information beyond that provided by the engine interface.

- Provides plan performance guarantees that are *individually* applicable to queries in the selectivity space.
- Considers only the *parametric optimal set of plans* (POSP) [12] as replacement candidates and therefore delivers, for errors that lie within the replacement plan's optimality region, robustness "for free". In contrast, the previously proposed algorithms in the literature may evaluate plans that are not optimal anywhere in the space.
- Is validated on *commercial* optimizers on both the classical TPC-H [26] and the recent TPC-DS [27] benchmarks.

We hasten to add that SEER, due to its non-intrusive design objective, only attempts to address selectivity errors that occur on the *base relations*, similar to [1]. However, since these base errors are often the source of poor plan choices due to the multiplier effect as they progress up the plan-tree [14], minimizing their impact could be of significant value in practical environments. Further, since SEER is a purely compile-time approach, it can be used in conjunction with run-time techniques such as adaptive query processing [9] for addressing selectivity errors in the higher nodes of the plan tree.

SEER is based on the *anorexic reduction of plan diagrams*.

Example. We now show an example of how anorexic reduction helps to identify selectivity-error-resistant plans: In Figure 1.2(a), estimated selectivities of around (14%,1%) lead to a choice of plan P70. However, if the actual selectivities at runtime turn out to be significantly different, say (50%,40%), executing with 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 1.2(b), 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 the run-time location, plan P1 has a cost of 135, while P70's cost of 402 is about *three times* more expensive.

It is easy to see, as in the above example, that the replacement plan will, *by definition*, be a robust choice for errors that lie within its optimality region, i.e. its *"endo-optimal"* region. This is the advantage, mentioned earlier, of considering replacements only from the POSP set of plans. The obvious question then is whether the sizes of these regions are typically large enough to materially improve the system performance.

A second, and even more important question, is: What if the errors are such that the run-time locations are *"exo-optimal"* w.r.t. the replacement plan? For example, if the run-time location happens to be at (80%,90%), which is outside the optimality region of P1? In this situation, nothing can be said upfront – the replacement could be much better, similar or much worse than the original plan. Therefore, ideally speaking, we would like to have a mechanism through which one could assess whether a replacement is *globally safe* over the entire parameter space.

1.3.3 Contributions and Results

We address the above problem from both theoretical and empirical perspectives. Through extensive experimentation with a representative suite of multi-dimensional TPC-H and TPC-DS-based query templates on leading commercial optimizers, we show that *plan diagram reduction typically produces plan choices that substantially curtail the adverse effects of selectivity estimation errors*. Therefore, it clearly has potential to improve performance in general, for both the endo-optimal and exo-optimal regions.

However, we have also encountered occasional situations where a replacement plan performs much worse in its exo-optimal region than the original optimizer choice, highlighting the need to establish an efficient criterion of when a specific swallowing is globally safe. To achieve this objective, we present a generalized mathematical model of the behavior of plan cost functions over the selectivity space. The model, although simple, is sufficient to capture the cost behavior of all plans that have arisen from our query templates. Using this model, we then prove that checks on only the *perimeter* of the selectivity space are sufficient to decide the safety of reduction over the entire space. These checks involve the costing of *"foreign plans"*, that is, of costing plans in their exo-optimal regions, a feature that has become available in the

current versions of several industrial-strength optimizers, including DB2[28] (Optimization Profile), SQL Server[30] (XML Plan) and Sybase[29] (Abstract Plan).

Apart from providing reduction safety, foreign-plan costing is additionally leveraged to both (a) enhance the reduction levels of the plan diagram, and (b) improve the complexity characteristics of the reduction process, as compared to our earlier CostGreedy reduction algorithm [11]. Note that the increased diagram reduction automatically implies *larger within-λ-of-optimal regions* for the retained plans, upfront guaranteeing more robustness.

In summary, we provide SEER, an efficient, effective and safe mechanism for identifying robust plans that are resistant, as compared to the optimizer's original choices, to errors in the base relation selectivity estimates. Through a detailed study of benchmark-based query templates on commercial optimizers, we empirically demonstrate that SEER provides robust good performance for industrial-strength database environments.

1.4 Organization

The remainder of this thesis is organized as follows: The SIGHT and ISIGHT algorithms for generating plan diagrams are presented in Chapter 2. The SEER reduction algorithm is discussed in Chapter 3. Our experimental framework and performance results are highlighted in Chapter 4. Finally, in Chapter 5, we summarize our conclusions and outline future research avenues.

Chapter 2

Efficient Production of Plan Diagrams

In this chapter we present the SIGHT algorithm, which can be used to efficiently generate *completely accurate* plan diagrams. Subsequently, we provide an inexact variant, the ISIGHT algorithm, which trades error, based on the user's bound, for reduction in optimization effort. Both algorithms require the cost-based optimizer to provide the following features:

Optimal Plan (OP): This feature, found in virtually every enterprise database product, provides the optimal plan (OP), as determined by the optimizer, for a given query.

Foreign Plan Costing (FPC): The "foreign plan costing" (FPC) feature provides an option for costing plans *outside* their native optimality regions. Specifically, the feature supports the "what-if" question: "What is the estimated cost of sub-optimal plan *p* if utilized at query location *q*?". FPC has become available in the current versions of several industrial-strength optimizers, including DB2 [28] (Optimization Profile), SQL Server [30] (XML Plan), and Sybase [29] (Abstract Plan).

Plan Rank List (PRL): The optimizer should support an API that provides not just the best plan but a "plan-rank-list" (PRL), enumerating the top k plans for the query. For example, with $k = 2$, both the best plan and the second-best plan are obtained when the optimizer is invoked on a query. However, to our knowledge, it is not yet available in any of the current systems. Therefore, we showcase its utility through our own implementation in a public-domain optimizer.

Specifically, it is assumed that for each query point, the optimizer provides both the best plan and the second-best plan, and an option to cost the second-best plan at other points in the selectivity space.

2.1 Obtaining the Second Best Plan

As mentioned in the Introduction, current optimizers use a variant of the algorithm used by the classical System R optimizer [22]. This algorithm uses the dynamic programming strategy to identify the optimal plan for a given query from an exponential (on the number of relations taking part in the query) search space.

This search space can be represented as a search tree, with the root node representing a subset of plans applicable for the given query. In the final step of the algorithm, the optimal plan is identified as the cheapest plan among this set of candidate plans. At first glance, we might mistake the second best plan from this set to be the global second best plan. This need not always be true, since there are multiple ways certain operations can be performed (eg. either a table-scan or an index-scan can be used to scan base tables), and it is possible to obtain a better plan by using a different choice for such operations in the optimal plan.

In order to find the global second best plan, we only need to search the restricted search space obtained by the path in the search tree taken for the optimal plan and compare the second-best plan in this space with the second-best candidate plan of the original dynamic programming exercise. This can be accomplished with a simple modification to the original dynamic programming algorithm, where, as we progress through the levels of the search tree, instead of choosing only the optimal sub-plan, we choose both the optimal and the second-best sub-plan, to be processed in the next step of the algorithm.

Approximate second-best plan: We have observed that second-best candidate plan obtained from the original dynamic programming exercise can be used as an effective approximation for the global second-best plan. This is *trivially implementable* in current optimizers since the candidate set is constructed by default during the optimization process.

2.2 Plan Diagrams

The selectivity space S is represented by a grid of points where each point $q(x, y)$ corresponds to a unique query with selectivities x, y in the X and Y dimensions, respectively. In a plan diagram P, generated for a query template Q, each $q \in$ P is associated with an optimal (as determined by the optimizer) plan P_i, and a cost $c_i(q)$ representing the estimated effort to execute q with plan P_i. 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. As mentioned earlier, the plan diagram is essentially a visual characterization of the parametric optimal set of plans (POSP) [12]. We use P and S interchangeably in the remainder of the thesis based on the context.

2.3 The SIGHT Algorithm

The SIGHT algorithm for a 2D query template is shown in Figure 2.1. The algorithm starts with optimizing the query point $q(x_{min}, y_{min})$ corresponding to the bottom-left query point in the plan diagram. Let p_1 be the optimizer-estimated optimal plan at q, with cost $c_1(q)$, and let p_2 be the *second best* plan, with cost $c_2(q)$. We then assign the plan p_1 to all points q' in the *first quadrant* relative to q as the origin, which obey the constraint that $c_1(q') \leq c_2(q)$. After this step is complete, we then move to the next unassigned point in row-major order relative to q, and repeat the process, which continues until no unassigned points remain.

This algorithm is predicated on the *Plan Cost Motonicity* (PCM) assumption that the cost of a plan is monotonically non-decreasing throughout the selectivity space, which is true in practice for most query templates [11].

The following theorem proves that the SIGHT algorithm will exactly produce the true plan diagram P without any approximation whatsoever. That is, *by definition*, there is zero plan-identity and plan-location errors.

THEOREM 2.1. *The plan assigned by SIGHT to any point in the approximate plan diagram* A *is exactly the same as that assigned in* P.

Proof. Let $P_o \subseteq$ P be the set of points which were optimized. Consider a point $q' \in$ P $\setminus P_o$ with a plan p_1. Let $q \in P_o$ be the point that was optimized when q' was assigned the plan p_1.

SIGHT (QueryTemplate QT)

1. Let A be an empty plan diagram.
2. Set $q = (x_{min}, y_{min})$
3. while ($q \neq null$)
   (a) Optimize query template QT at point q.
   (b) Let p_1 and p_2 be the optimal and second-best plan at q, respectively.
   (c) for all unassigned points q' in the first quadrant of q
   if ($c_1(q') \leq c_2(q)$), assign plan p_1 to q'
   (d) Set q = next unassigned query point in A
4. Return A
5. End Algorithm SIGHT

Figure 2.1: The SIGHT Algorithm

Let p_2 be the second best plan at q.

For the sake of contradiction, let p_k $(k \neq 1)$, be the optimal plan at q'. We know that for a cost-based optimizer, $c_k(q') < c_1(q')$. This implies that $c_k(q') < c_2(q)$ (due to the algorithm). Using the PCM property, we have $c_k(q) \leq c_k(q') \Rightarrow c_1(q) \leq c_k(q) < c_2(q)$. This means that p_2 is not the second best plan at q, a contradiction. □

2.3.1 Handling non-PCM templates

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 costs will still remain monotonic in an appropriate quadrant [11], as shown in Table 2.1 for the 2D case.

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

Table 2.1: PCM Behaviour

The algorithm can be easily modified to take the appropriate quadrant into consideration. For example, if the costs are monotonically non-decreasing along the third quadrant, then the algorithm starts processing from the top-right of the plan diagram (Step 2),and the plan assignment is performed along the third quadrant (Step 3c). The quadrant in which the cost of a plan is non-decreasing can be easily obtained by comparing the costs of the plan at the 4 corners of the selectivity space.

2.4 The ISIGHT Algorithm

While SIGHT always gives zero error, we now investigate the possibility of whether it is possible to utilize the permissible error bound of ϵ given by the user to reduce the computational overheads of SIGHT. To this end, we propose the following ISIGHT algorithm: The plan assignment constraint $c_i(q') \leq c_j(q)$ is relaxed to be $c_i(q') \leq (1+\delta)c_j(q)$ with $(\delta > 0)$, resulting in fewer optimizations being required to fully assign plans in the diagram. The choice of δ is a function of the user's ϵ error bound and and μ_i, the slope of the cost function c_i at q. Our empirical assessment indicates that setting $\delta = 0.1 * \mu_i * \epsilon$ (e.g. with $\epsilon = 10\%$ and $\mu_i = 1$, $\delta = 0.01$) is sufficient to both meet the error requirements and simultaneously significantly reduce the overheads. For example, $\epsilon = 10\%$ can be achieved with only around **1%** overheads.

Chapter 3

Identifying Robust Plans through Plan Diagram Reduction

We now turn our attention to the problem of obtaining plans that are resilient to selectivity errors. For ease of exposition, we assume in the following discussion that the SQL query template is 2-dimensional in its selectivity variations – the extension to higher dimensions is straightforward.

3.1 Problem Framework

3.1.1 Reduced Plan Diagrams

The **Plan Diagram Reduction** problem is defined as follows [11]: Given an input plan diagram P, and a maximum-cost-increase threshold λ ($\lambda \geq 0$), find a reduced plan diagram R with *minimum cardinality* such that for every plan P_i in P,

1. Either $P_i \in$ R, or
2. $\forall\, q \in P_i$, the assigned replacement plan $P_j \in$ R guarantees $\frac{c_j(q)}{c_i(q)} \leq (1+\lambda)$

That is, find the maximum possible subset of the plans in P that can be completely "swallowed" by their sibling plans in the POSP set. A point worth reemphasizing here is that the threshold

constraint applies on an *individual query* basis. For example, setting $\lambda = 10\%$ stipulates that the cost of *each* query point in the reduced diagram is within 1.1 times its original value.

It was proved in [11] that the above problem is NP-Hard. Therefore, an efficient heuristic-based online algorithm, called **CostGreedy**, was proposed and shown to deliver near-optimal "anorexic" levels of reduction, wherein the plan cardinality of the reduced diagram usually came down to around 10 or less for a λ-threshold of only 20%. In a nutshell, complex plan diagrams can be easily made very simple without materially affecting the query processing quality.

3.1.2 Selectivity Estimation Errors

Consider a specific query point q_e, whose optimizer-estimated location in S is (x_e, y_e). Denote the optimizer's optimal plan choice at point q_e by P_{oe}. Due to errors in the selectivity estimates, the *actual* location of q_e could be different at execution-time – denote this location by $q_a(x_a, y_a)$, and the optimizer's optimal plan choice at q_a by P_{oa}. Assume that P_{oe} has been swallowed by a sibling plan during the reduction process and denote the replacement plan assigned to q_e in R by P_{re}. Finally, extend the definition of query cost (which applied to the optimal plan) to have $c_i(t)$ denote the cost of an arbitrary POSP plan P_i at an arbitrary query point t in S.

With respect to R, the actual query point q_a will be located in one of the following disjoint regions of P_{re} that together cover S:

Endo-optimal region of P_{re}: Here, q_a is located in the optimality region of the replacement plan P_{re}, which also implies that $P_{re} \equiv P_{oa}$. Since $c_{re}(q_a) \equiv c_{oa}(q_a)$, it follows that the cost of P_{re} at q_a, $c_{re}(q_a) < c_{oe}(q_a)$ (by definition of a cost-based optimizer). Therefore, improved resistance to selectivity errors is always *guaranteed* in this region.

Swallow-region of P_{re}: Here, q_a is located in the region "swallowed" by P_{re} during the reduction process. Due to the λ-threshold constraint, we are assured that $c_{re}(q_a) \leq (1 + \lambda)c_{oa}(q_a)$, and by implication that $c_{re}(q_a) \leq (1 + \lambda)c_{oe}(q_a)$. Now, there are two possibilities: If $c_{re}(q_a) < c_{oe}(q_a)$, then the replacement plan is again guaranteed to improve the

resistance to selectivity errors. On the other hand, if $c_{oe}(q_a) \leq c_{re}(q_a) \leq (1+\lambda)c_{oe}(q_a)$, the replacement is guaranteed to not cause any real harm, given the small values of λ that we consider in this thesis.

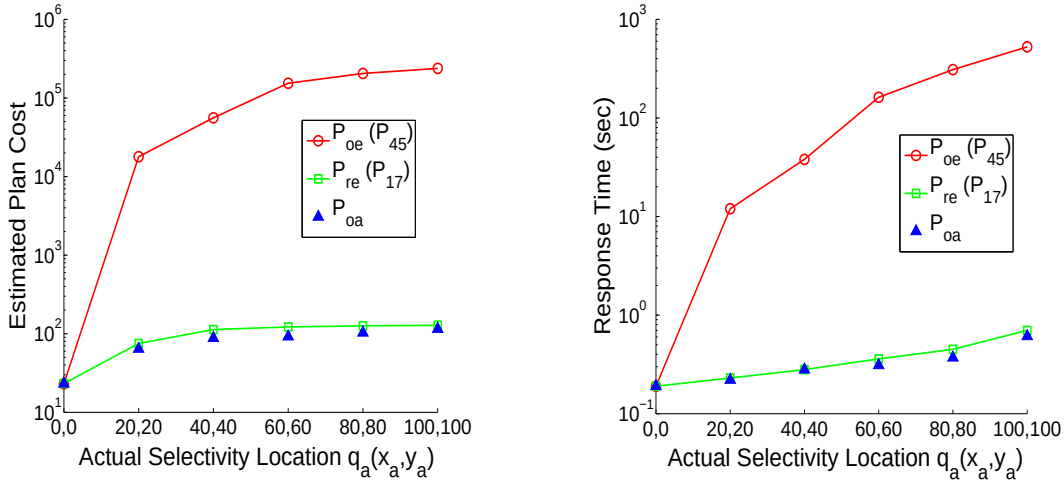
Exo-optimal region of P_{re}: Here, q_a is located outside both the endo-optimal and swallow-regions of P_{re}. At such locations, we cannot apriori predict P_{re}'s behavior, and therefore the replacement may not always be a good choice – in principle, it could be *arbitrarily worse*. Therefore, we would like to ensure that even if the replacement does not provide any improvement, it is at least guaranteed to not do any harm. That is, the *exo-optimal region should have the same performance guarantees as the swallow-region*. We show in Section 3.2 how this objective can be efficiently achieved through simple but powerful checks to decide when replacement is advisable.

3.1.3 Motivational Scenarios

Given the above framework, we now present example scenarios to motivate (a) the error-resistance utility of plan diagram reduction, and (b) the need for safety in this process.

Our first scenario, typical of that seen in most of our experiments, demonstrates how the replacement plan P_{re} can provide extremely substantial improvements *throughout the selectivity space*. Specifically, on a vanilla PC with a popular commercial optimizer, we generated a plan diagram for a query template based on TPC-H Q5, with selectivity variations on the CUSTOMER and SUPPLIER relations, and carried out reduction with $\lambda = 10\%$. For this diagram, with $q_e = (0.36, 0.05)$, and a representative set of actual locations (q_a) along the principal diagonal of S, the costs of P_{oe} (P45), P_{re} (P17) and P_{oa} (the optimal plan at each q_a location) are shown in Figure 3.1(a) – note that the costs are measured on a *log scale*.

It is clear from Figure 3.1(a) that the replacement plan P_{re} provides *orders-of-magnitude* benefit w.r.t. P_{oe}. In fact, the error-resistance is to the extent that it virtually provides "immunity" to the error since the performance of P_{re} is close to that of the *locally optimal plan* P_{oa} throughout the space, although the endo-optimal region of P_{re} constitutes only a very small fraction of this space.



(a) Compile-Time (b) Run-Time

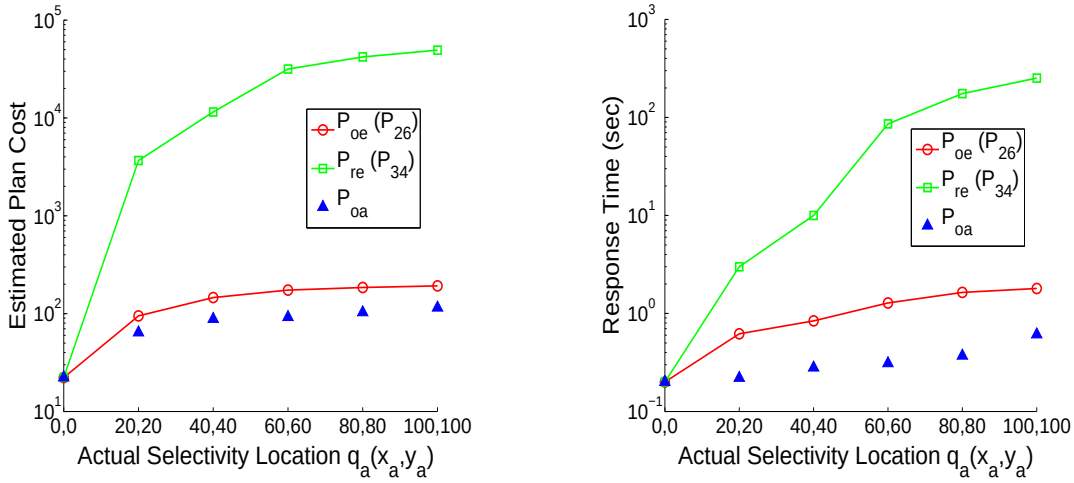
Figure 3.1: Beneficial Impact of Plan Replacement

To demonstrate that the benefits anticipated from the compile-time analysis do translate to corresponding improvements *at runtime*, we show in Figure 3.1(b) the query response times (again measured on a *log scale*) of P_{oe} (P45), P_{re} (P17) and P_{oa} at the same q_a locations. It is vividly clear in this picture that huge savings in processing time are obtained by using the replacement plan instead of the optimizer's original choice, and that the replacement's performance is virtually indistinguishable from the optimal choices.

While performance improvements are usually the order of the day, there are occasional situations wherein P_{re} performs worse than P_{oe} at q_a. A particularly egregious example, arising from the *same* plan diagram described above, is shown in Figure 3.2(a) for $q_e = (0.03, 0.14)$ – we see here that it is now the replacement plan P_{re} (P34), which is *orders-of-magnitude* worse than P_{oe} (P26) in the presence of selectivity errors. This compile-time assessment is corroborated in Figure 3.2(b) which shows the corresponding query response times.

3.1.4 Robust Reduction

From the above discussion, it is clear that we need to ensure that only safe replacements are permitted. This means that replacement should be permitted only if the λ threshold criterion



(a) Compile-Time (b) Run-Time

Figure 3.2: Adverse Impact of Plan Replacement

is satisfied not just at the estimated point, but *at all locations* in the selectivity space. At the same time, it is important to ensure that the safety check is not unnecessarily conservative, preventing most plan replacements, and in the process losing all the error-resistance benefits. Therefore, the overall goal is to maximize plan diagram reduction without violating safety considerations. More formally, our problem formulation is:

Robust Reduction Problem. Given an input plan diagram P, and a maximum cost-increase-threshold λ ($\lambda \geq 0$), find a reduced plan diagram R with *minimum plan cardinality* such that for every plan P_i in P,

1. $P_i \in$ R, or
2. $\forall\ q \in P_i$, the assigned replacement plan $P_j \in R$ guarantees $\forall$ query points $q' \in$ P, $\frac{c_j(q')}{c_i(q')} \leq (1+\lambda)$

That is, find the minimum-sized error-resistant "cover" of plans that reduces the plan diagram P without increasing the cost of any reassigned query point by more than the cost increase threshold, *irrespective of the actual location of the query at run-time*.

It is easy to see that the Robust Reduction problem is NP-Hard, just like the standard Plan Diagram Reduction problem, and therefore we present a heuristic-based algorithm later in

Section 3.3. But, prior to that, we show in the following section how replacement safety can be checked efficiently.

3.2 Ensuring Robust Reduction

To find an error-resistant cover of the plan diagram, we need to evaluate the behavior of each replacement plan P_{re}, w.r.t. its swallowing target P_{oe}, at *all points* in S. This requires, in principle, finding the costs of P_{oe} and all potential P_{re} at every point in the diagram. Of course, P_{oe} and P_{re} need not be costed in their respective *endo-optimal* regions, since these values are already known through the plan diagram production process. The remaining *exo-optimal* costs can be obtained using the FPC feature, that is now supported in several industrial-strength optimizers, as mentioned in the Introduction.

While the above solution is conceptually feasible, it is practically unviable due to its enormous computational overheads. Plan-costing is certainly cheaper than the optimizer's standard optimal-plan-searching process [13], but the overall overhead is still $O(nm)$ where n and m are the number of plans and the number of points, respectively, in P. Typical values of n range from the several tens to several hundreds, while m is of the order of several thousands to several hundreds of thousands, making an exhaustive approach impractical.

The above situation motivates us to study whether it is possible, based on using FPC at only a few select locations, to *infer* the behavior in the rest of the space. In the remainder of this section, we describe our strategy for making such an inference. We begin by designing a parameterized mathematical model for characterizing plan cost behavior. Our model is grossly simplified in comparison to those used in real optimizers, which are much more complex [19, 18]. However, what we have found in practice (with several hundred distinct plans arising out of TPC-H and TPC-DS-based query templates on industrial optimizers) is that with appropriate settings of the parameters, our simple model is quite accurate, both behaviorally and quantitatively. The reasons are that (a) in our problem space all parameters, barring the selectivities, are *constant*, resulting in complex models degenerating to comparatively simple equivalents; (b) we are *fitting* the model to the observed cost behaviors, rather than trying to

predict them; and (c) our modeling is at the level of entire plans, aggregating the effects of several individual operators, thereby reducing the variability. Moreover, the quantitative accuracy is a bonus – it is not really required since only *behavioral* accuracy is necessary for our scheme to work.

3.2.1 Modeling Plan Cost Functions

For ease of presentation, we will initially assume that our objective is to model the cost behavior of plans with respect to a 2-D selectivity space (e.g. Figure 1.2(a)) corresponding to distinct relations R_x and R_y. The extension to higher dimensions is straightforward and is provided later in this section.

In current optimizers, the operators in the execution plan are all typically either *unary* or *binary* with regard to their inputs. Therefore, given a specific plan operator tree, like the sample one shown in Figure 3.3 (obtained on a $R_x = \text{LINEITEM}$, $R_y = \text{CUSTOMER}$ selectivity space), we can define the following types of nodes:

Selectivity Nodes: These are the unary nodes that implement the selection operations on relations R_x and R_y. In Figure 3.3, they are colored orange, corresponding to Index Scans on the LINEITEM and CUSTOMER relations, respectively.

Dependent Nodes: These are the nodes in the tree that have at least one Selectivity Node in the sub-tree below them. They are colored blue in Figure 3.3.

Independent Nodes: These are all the remaining nodes in the tree that do not belong to either of the above two categories. They are colored white in Figure 3.3.

3.2.2 Node Cost Models

We now enumerate the cost models that can be associated with the above node categories on the 2-D selectivity space S. Our formulation is based on detailed observations of cost behavior of individual operators on commercial database optimizers. In the following, the variables x and y are used to denote the (fractional) selectivities on the respective dimensions.

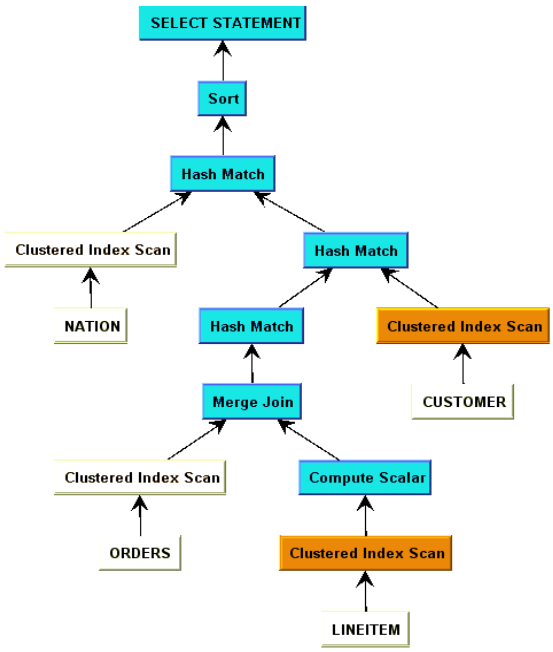


Figure 3.3: Sample Plan Tree

Independent Nodes: Since these nodes do not have a Selectivity Node in their sub-tree, variations in x and y do not change their inputs, and consequently their outputs. Therefore, for a given plan, the costs at these nodes remain the same throughout S.

Selectivity Nodes: The input cardinalities for these nodes will be constant (equal to the corresponding base relation's cardinality n) while the output cardinality is directly dependent on the selectivity value. Therefore, the cost behavior can be captured by the simple linear model involving coefficients a_1 and a_2 shown in Table 3.1. For example, *Table-Scans* will have $a_1 = 0$, while *Index-Scans* are likely to have non-zero values for both constants.

Dependent Unary Nodes: The input cardinalities for these nodes will be a function of x and/or y, and the associated family of cost models is as shown in Table 3.1. For operators such as *Aggregates*, *Arithmetic Expressions*, *Scalar functions*, etc. the simple linear model will apply, whereas the logarithmic model would apply to operators such as *Sort* and *Group By* that require multiple passes over the data.

Dependent Binary Nodes: These are the nodes that represent binary set operators such as

Node Type	Input Cardinalities		Cost Model
Selectivity Node ($\sigma = x$)	n		$a_1 nx + a_2$
Dependant Unary Nodes	$n_1 x$		$a_1 n_1 x + a_2$
			$a_1 n_1 x \log n_1 x + a_2$
	$n_1 xy$		$a_1 n_1 xy + a_2$
			$a_1 n_1 xy \log n_1 xy + a_2$
Dependant Binary Nodes	$n_1 x$	n_2	$a_1 n_1 x + a_2 n_2 + a_3 n_1 n_2 x + a_4$
	$n_1 xy$	n_2	$a_1 n_1 xy + a_2 n_2 + a_3 n_1 n_2 xy + a_4$
	$n_1 x$	$n_2 y$	$a_1 n_1 x + a_2 n_2 y + a_3 n_1 n_2 xy + a_4$

Table 3.1: Cost Models for Various Node Types

Join, *Union*, *Minus*, etc. The different types of input possibilities and the associated cost models are shown in Table 3.1.

Note that we deliberately do not consider the case where *both* the inputs to the binary node are functions of x (or y or xy). This is because it is easy to prove that such a situation is not possible unless operators have *binary outputs* – we have not encountered any such operators in our study.

LEMMA 3.1. *There cannot be a binary node that has both inputs to be functions of x (or functions of y, or functions of xy).*

Proof. If there exists a binary node N with input cardinalities $n_1 x$ and $n_2 x$, then there should exist some node in its subtree that has a *binary output*. However, we know that all nodes in the plan tree have unary outputs (since there is no cycle in the tree). A similar argument holds for the y and xy cases. □

Cost Model of a Complete Plan

The cost function of the entire plan is the aggregate sum of the costs of the individual nodes. Considering all possible cost models a node can have, we can conclude that the overall cost model of a plan for a 2D selectivity space is of the form

$$\begin{aligned} Cost(x, y) =& a_1 x + a_2 y + a_3 xy + a_4 x \log x + a_5 y \log y + \\ & a_6 xy \log xy + a_7 \end{aligned} \tag{3.1}$$

where $a_1, a_2, a_3, a_4, a_5, a_6, a_7$ are coefficients, and x, y represent the selectivities of R_x and R_y, respectively.

Modeling a specific plan requires suitably choosing the seven coefficients, and this is achieved through standard surface-fitting techniques, described in Section 4.

3.2.3 Extension to d-dimensional spaces

Generalizing the arguments used in the 2D case, we obtain the following cost model for a d-dimensional selectivity space.

$$Cost(x_1, ..., x_d) = \sum_{i_1}(a_{i_1}x_{i_1} + b_{i_1}x_{i_1}\log x_{i_1}) + \sum_{i_1<i_2}(a_{i_1i_2}x_{i_1}x_{i_2} + b_{i_1i_2}x_{i_1}x_{i_2}\log x_{i_1}x_{i_2}) + ... + a_{12..d}(x_1x_2x_3..x_d) + b_{12..d}(x_1x_2x_3..x_d)\log(x_1x_2x_3..x_d) + a_0 \quad (3.2)$$

where the a's and b's are the coefficients and the $x_i, i = 1...d$ represent the d relational selectivities.

3.2.4 Replacement Safety Conditions

For the 2D scenario, using the above 7-coefficient cost model, our goal now is to come up with an efficient mechanism to assess, given an optimal plan P_{oe}, candidate replacement plan P_{re} and a cost-increase threshold λ, whether it would be safe from a *global* perspective to have P_{re} swallow P_{oe}.

Let the cost functions for P_{re} and P_{oe} be

$$f_{re}(x, y) = a_1x + a_2y + a_3xy + a_4x\log x + a_5y\log y + a_6xy\log xy + a_7 \quad (3.3)$$

and

$$f_{oe}(x, y) = b_1x + b_2y + b_3xy + b_4x\log x + b_5y\log y + b_6xy\log xy + b_7 \quad (3.4)$$

respectively. Now consider the **"safety function"**

$$f(x, y) = f_{re} - (1 + \lambda) f_{oe} \tag{3.5}$$

which captures the differences between the costs of P_{re} and a λ-inflated version of P_{oe} in the selectivity space. All points where $f(x, y) \leq 0$ are referred to as *SafePoints* whereas points that have $f(x, y) > 0$ are called *ViolatingPoints*. For a replacement to be globally safe, there should be no ViolatingPoint anywhere in the selectivity space.

In the following, we will use LR-Boundaries to collectively denote the left and right boundaries of the selectivity space, and TB-Boundaries to collectively denote the top and bottom boundaries of the space.

For a specific value of y, the safety function $f(x, y)$ can be rewritten as

$$f_y(x) = g_1 * x + g_2 * x \log x + g_3$$

for appropriate coefficients g_1, g_2, g_3. Similarly, we can define $f_x(y)$. With this terminology, the following theorem provides us with conditions for checking whether the selectivity space is safe for the plan-pair (P_{oe},P_{re}) with regard to replacement.

THEOREM 3.2. *For a plan-pair (P_{oe},P_{re}) and a selectivity space* S *with corners* $[(x_1, y_1)$, (x_1, y_2), (x_2, y_2), $(x_2, y_1)]$, *the replacement is safe (i.e., within λ-threshold) in* S *if any one of the conditions, SC1 through SC6, given in Table 3.2 is satisfied.*

	Left Boundary	**Right Boundary**	**Top Boundary**	**Bottom Boundary**
SC1	Safe	Safe	$f''_{y_2}(x) \geq 0$	$f''_{y_1}(x) \geq 0$
SC2	$f'_y(x_1) \leq 0$ & Safe	Safe	$f''_{y_2}(x) < 0$	$f''_{y_1}(x) < 0$
SC3	Safe	$f'_y(x_2) \geq 0$ & Safe	$f''_{y_2}(x) < 0$	$f''_{y_1}(x) < 0$
SC4	$f''_{x_1}(y) \geq 0$	$f''_{x_2}(y) \geq 0$	Safe	Safe
SC5	$f''_{x_1}(y) < 0$	$f''_{x_2}(y) < 0$	$f'_x(y_2) \geq 0$ & Safe	Safe
SC6	$f''_{x_1}(y) < 0$	$f''_{x_2}(y) < 0$	Safe	$f'_x(y_1) \leq 0$ & Safe

Table 3.2: Safety Satisfaction Conditions

In order to prove the above theorem, we will start with deriving two lemmas – the first provides us with a condition that is sufficient to ensure safety of all points on the straight line segment joining a pair of safe points, while the second describes the behaviour of the slope of the safety function.

LEMMA 3.3 (Line Safety). *Given a fixed $y = y_o$, and a pair of safe points (x_1, y_o) and (x_2, y_o) with $x_2 > x_1$, the straight line joining the two points is safe if the slope $f'_{y_o}(x)$ is either*

(i) *monotonically non-decreasing, OR*

(ii) *monotonically decreasing with $f'_{y_o}(x_1) \leq 0$ or $f'_{y_o}(x_2) \geq 0$*

A similar result holds when x is fixed.

Proof. The various possible behaviors of $f_y(x)$ are shown in Figure 3.4 as Curves (a) through (e). When the slope $f'_{y_o}(x)$ is monotonically non-decreasing (i.e. Condition (i) is satisfied), the safety function curve that connects the two safe points is guaranteed to lie *below* the straight line joining the two points – Curve (a) in Figure 3.4 shows an example of this situation. This ensures that the safety function along the given line segment is always negative and hence safe.

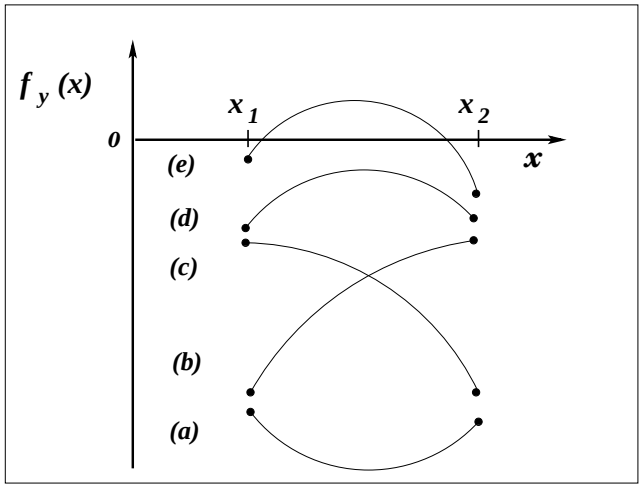


Figure 3.4: Behavior of the safety function $f_y(x)$

If, on the other hand, $f'_{x_o}(y)$ is monotonically decreasing, then the possible behaviors of the safety function $f_{x_o}(y)$ are shown in Curves (b) through (e) in Figure 3.4. Curves (b) and

(c) denote the behaviour of the safety function when Condition (ii) is satisfied, and clearly the value of the safety function is again negative in the given range. □

In Figure 3.4, Curve (d) also corresponds to a safe scenario – however, it is not possible to differentiate between Curve (d) and the unsafe case, namely Curve (e), without explicitly computing the safety function at every point on the given line-segment. Hence, we *conservatively* categorize both cases as unsafe. We have also observed that the case corresponding to Curve (e) occurs rarely in practice.

LEMMA 3.4 (Slope Behavior). *If the slope of the safety function, $f'_y(x)$, is non-decreasing (resp. decreasing) along the line-segments $y = y_1$ and $y = y_2$, then it is non-decreasing (resp. decreasing) for all line segments in the interval (y_1, y_2). A similar result holds for $f'_x(y)$.*

Proof. Consider the slope of the safety function

$$f'_{y_o}(x) = \frac{df_{y_o}(x)}{dx} = g_1 + g_2(1 + \log x) \tag{3.6}$$

For $x \in (0, 1)$, this slope is monotonic and its behavior depends on the sign of g_2. From Equations 3.3 and 3.4, we know that g_2 can be written as the following function of y

$$\begin{aligned} g_2(y) &= (a_4 - (1+\lambda)b_4) + (a_6 - (1+\lambda)b_6)y \\ &= (k_1 + k_2 y) \end{aligned} \tag{3.7}$$

where k_1 and k_2 are constants.

Since $g_2(y)$ is a linear function of y, the Lemma immediately follows. □

We now prove Theorem 3.2 using the LineSafety and SlopeBehavior lemmas:

Proof. Consider the SC1 condition in Table 3.2: Since $f''_y(x) \geq 0$ (i.e. slope $f'_y(x)$ is non-decreasing) at the TB-boundaries, then from Lemma 3.4, we know that the slope $f'_y(x)$ is non-decreasing throughout the range (y_1, y_2).

Moving on to the SC2 and SC3 conditions: Since $f''_y(x) < 0$ (i.e. slope $f'_y(x)$ is decreasing) at the TB-boundaries, then from Lemma 3.4, we know that the slope $f'_y(x)$ is decreasing throughout the range (y_1, y_2). Further, we know that for a given $y = y_o \in (y_1, y_2)$, either $f'_{y_o}(x_1) \leq 0$ (SC2) or $f'_{y_o}(x_2) \geq 0$ (SC3).

Thus, when SC1, SC2 or SC3 is satisfied, then for all lines between points (x_1, y) and (x_2, y), $y \in (y_1, y_2)$, the end-points are safe (because the LR-boundaries are safe), and the slope conditions given in Lemma 3.3 are satisfied. Hence, all such line-segments are safe, the union of which is the given region.

Similar arguments can be used to show safety of the region when conditions SC4, SC5 or SC6 are satisfied. Hence the theorem. □

The test criteria of Theorem 3.2 are utilized for determining reduction safety in the SafetyCheck algorithm, described next. A related point to note here is that these checks are *conservative* in that it is possible to have global safety even if none of the conditions are met – i.e. the test is sufficient, but not necessary.

3.3 The SEER Algorithm

In this section, we first describe the safety checking procedure, which given a plan-pair (P_{oe}, P_{re}), responds whether the replacement of P_{oe} by P_{re} is globally safe throughout the selectivity space S. We then present and analyze the SEER algorithm which uses this procedure to perform error-resistant plan diagram reduction.

In the following, we will assume that the selectivity space S is represented by a grid G, with $m = r \times r$ points, i.e. the grid resolution in each dimension is r.

3.3.1 Safety Checking

To implement safe reduction in a 2-D plan diagram, we need to be able to check for the satisfaction of any of the conditions (SC1 through SC6) stipulated in Theorem 3.2. A straightforward way to achieve this is the following *Perimeter Test* procedure:

Perimeter Test. First compute the safety function at all points on the *perimeter* of G – this is obtained through the foreign-plan-costing (FPC) feature. Then, compute the slope behavior (non-decreasing or decreasing) along all the grid lines – this is achieved by evaluating the slopes at the matching end-points on the perimeter and comparing the values. The slope at a perimeter point is approximated by computing the value of the safety function at its immediate

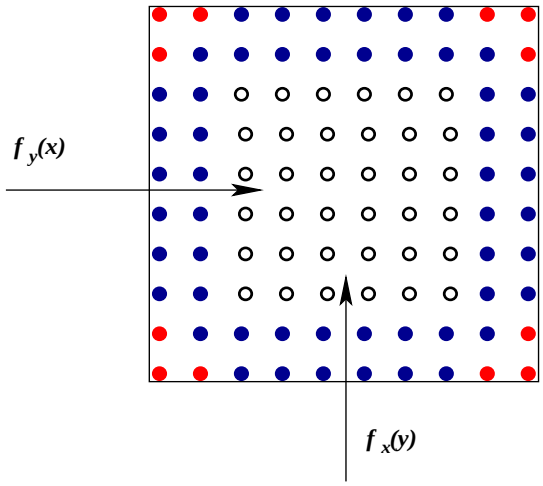


Figure 3.5: Perimeter and Wedge Test

internal neighbor – i.e., along the "inner perimeter", and evaluating the slope of the line segment joining these two points. Finally, use these results to verify whether any of the 6 safety conditions are satisfied.

In the Perimeter test, the number of FPC operations is $2 * 4(r - 1)$ for the perimeter (the 2 is due to having to compute both f_{re} and f_{oe}), while the computation of the slopes takes an additional $2 * 4(r - 3)$ costings of the inner perimeter, leading to a total of approximately $16r$. Note that this is much less than the $2r^2$ FPC operations required by a brute-force approach of costing both plans at all points in the diagram. For example, with $r = 100$, the overhead is brought down by over an order of magnitude. The red and blue points shown in Figure 3.5 are to be costed in this test.

An obvious minor improvement that could be carried out on the $16r$ overhead is to perform the inner perimeter costings only when conditions SC1 and SC4 are violated. In this case, only one of SC2 or SC3 (resp. SC5 or SC6) can be valid. Hence, we need to perform FPC operations only at *two* boundaries of the inner perimeter, one along each dimension. This reduces the FPC overhead to $12r$.

Wedge Test. We now present a powerful optimization, called *Wedge Test*, that allows conditions SC1 and SC4 to be checked with a *constant* number of FPC, specifically 24, *irrespective of the resolution*. This is based on the observation that the slope of the safety function is

a monotonic function (Equation 3.6). Thus, by comparing the slopes at the corners of the space, we can infer the slope behaviour of the safety function along its boundaries. Applying Lemma 3.3, the safety of the boundaries can also be inferred. Hence, it is sufficient to perform FPC only at each corner of the space and its two adjacent points on the perimeter boundaries – that is, at the "corner wedges". Only the red points shown in Figure 3.5 are to be costed in this test.

Based on the above observations, we employ a two-stage process of safety-checking – in the first stage, use the extremely cheap Wedge Test check, and only if it fails, use the more expensive Perimeter Test to verify replacement safety.

Note that once a plan is costed at a given location, we store this cost in a cache for reuse later, ensuring no redundant computations.

3.3.2 Plan Diagram Reduction

We now show how the above safety checks are integrated into the SEER procedure for plan diagram reduction. Note that SEER's design is completely different from that of CostGreedy [11] because now reduction is permitted only if it satisfies a safety criterion that is applicable over S, whereas CostGreedy's attention is limited to only P_{oe}'s endo-optimal region.

The complete SEER algorithm is shown in Figure 3.6. Here, a Set-Cover instance is first created from the input plan diagram P. Then the two-stage global safety checking procedure of the Wedge Test, followed by the Perimeter Test, is implemented to evaluate replacement possibilities across each pair of plans in P, and the Set-Cover instance is updated accordingly. Finally, the resulting instance is solved using the standard greedy techniques [23, 10] to obtain the reduced plan diagram R.

Analysis. As discussed earlier, each replacement assessment of a plan-pair (P_{oe},P_{re}) requires atmost $O(r)$ FPCs to be performed. There are $O(n^2)$ such comparisons performed by the algorithm. However, since we cache the already obtained costs, the amortized number of FPC to be performed per plan is $O(r)$. Thus, for grid G with $m = r \times r$ points, the comparison of all plan pairs requires only $O(n\sqrt{m} + n^2)$ time. Solving the Set-Cover problem using the Greedy

SEER (Plan Diagram P, Threshold λ)

1. Create a Set-Cover Instance $I = (U, S)$, where $S = \{S_1, S_2, ..., S_n\}$, $U = \{1, 2, ..., n\}$, corresponding to the n plans in the original plan diagram P.
2. Set each $S_i = \{i\}, \forall i = 1...n$
3. For each pair of plans (P_i, P_j) do

   if (WEDGE_TEST (P_i,P_j,λ) == Safe) then

   $S_i = S_i \bigcup \{j\}$

   else if (PERIMETER_TEST (P_i,P_j,λ) == Safe) then

   $S_i = S_i \bigcup \{j\}$
4. Solve the Set-Cover instance I using the Greedy Set-Cover algorithm to identify the plans retained in R.

Figure 3.6: The SEER Reduction Algorithm

Set-Cover algorithm [23, 10] requires $O(n^2)$ time. This results in an $O(n\sqrt{m} + n^2)$ reduction algorithm. Further, since the set cover instance created has $|U| = n$, the approximation factor of this reduction algorithm is $O(\log n)$.

The above bounds and approximation factors for SEER compare very favorably with those of the CostGreedy reduction algorithm [11], which has time complexity $O(nm)$ and approximation factor of $O(\log m)$, since typically $n << m$.

3.3.3 Extension to Higher Dimensions

The SafetyCheck algorithm used to verify the safety of the replacement of P_{oe} by P_{re} in a d-dimensional selectivity space is given in Figure 3.7. The above algorithm recursively finds the safe area of the $(d-1)$-dimension "slices" of the input d-dimension selectivity space. When $d = 2$, the WEDGE_TEST and PERIMETER_TEST methods are used to check for safety. The SEER algorithm incorporating this checking mechanism is shown in Figure 3.8.

Each d-dimensional plan diagram is composed of r $(d-1)$-dimensional plan diagrams. The time complexity of the SafetyCheck algorithm for the base case when $d = 2$ is $O(r)$. Thus the SafetyCheck algorithm runs in $O(r^{d-1})$ time. Given a grid with $m = r^d$ points, FPC is performed $O(m^{\frac{d-1}{d}})$ times for each plan pair. Thus, the time complexity of the SEER reduction algorithm for a d-dimensional plan diagram is $O(m^{\frac{d-1}{d}} n + n^2)$.

SafetyCheck (Plan Diagram P, Threshold λ, Plan P_{oe}, Plan P_{re}, Dimension d)

1. if(d == 2)
   (a) if (WEDGE_TEST (P,P_i,P_j,λ) == Safe) then
   return $true$.
   else if (PERIMETER_TEST (P,P_i,P_j,λ) == Safe) then
   return $true$.
   (b) **return** $false$.
2. else
   (a) $safety = true$;
   (b) for each $(d-1)$-dimension slice P' of P
   $safety$ = $safety \wedge$ SafetyCheck (P',$lambda$, P_{oe}, P_{re}, $d-1$)
3. **return** $safety$.

Figure 3.7: n-Dimensional SafetyCheck Algorithm

SEER (Plan Diagram P, Threshold λ)

1. Create a Set-Cover Instance $I = (U, S)$, where $S = \{S_1, S_2, ..., S_n\}$, $U = \{1, 2, ..., n\}$, corresponding to the n plans in the original plan diagram P.
2. Set each $S_i = \{i\}, \forall i = 1...n$
3. For each pair of plans (P_i, P_j) do
   if (SafetyCheck (P,λ,P_i,P_j,d) == $true$) then
   $S_i = S_i \bigcup \{j\}$
4. Solve the set-cover instance I using the Greedy Setcover algorithm to identify the plans retained in R.

Figure 3.8: n-Dimensional SEER Reduction Algorithm

3.4 Variants of SEER

3.4.1 LiteSEER: A Fast Variant

The SEER design makes conscious efforts, as described above, to minimize the computational overheads, but these overheads do grow with increasing dimensionality of the query template. Therefore, we have also designed and evaluated LiteSEER, a light-weight heuristic-based algorithm that trades SEER's safety guarantee for providing rapid running-times. In LiteSEER, a replacement is simply assumed to be safe if *all the corner points of the selectivity space are safe*. The intuition behind this observation is that when two points are safe, then the straight line joining them is also usually safe. This is corroborated by our experimental results which indicate that the heuristic provides almost the same safety as that obtained through the strict-checking criteria of SEER.

Given a d-dimensional plan diagram P with n plans, the LiteSEER algorithm only computes the safety function at the 2^d corners of the associated selectivity space. It immediately follows that its overall complexity is $O(2^d n + n^2)$. Since, in most practical scenarios of interest, $2^d << n$ (e.g. in the 2-D case, $2^d = 4$, while n is typically in the several tens, if not more), the effective complexity turns out to be $O(n^2)$. Note that, in principle, in the absence of any apriori information, this is the *minimum work* required to be executed by any reduction algorithm.

3.4.2 PartialSeer

The problem formulation for robust reduction required the replacement plan to be *globally safe*. As a generalized variant, the safety criteria can be relaxed to allow a plan P_{re} to replace plan P_{oe} if P_{re} is safe in at least a user-defined *minimum safe fraction (MSF)* of the area covered by S ($MSF \leq 1$).

In order to assess partial safety, we first perform the WEDGE_TEST and PERIMETER_TEST checks for global safety. If this fails, we verify whether the slope criteria of any of the 6 conditions given in Theorem 3.2 is satisfied. If true, we allow plan P_{re} to replace plan P_{oe} if

1. At least two adjacent boundaries in the perimeter of S are safe; and

2. The MSF requirement is met in S.

The reason for restricting our attention to situations where at least two adjacent boundaries are safe is that, for this case, an efficient algorithm can be set up to check satisfaction of the area requirement, as described below. Figure 3.9 shows the modified SafetyCheck algorithm that finds the safe area when the left and bottom boundaries of S are safe. The algorithm is similar when other boundaries are safe.

From Theorem 3.2 we know that the safe (and violating) points form contiguous regions in S when the slope criteria of at least one of the size conditions are satisfied. Since the left and bottom boundaries of the grid are safe, the x and y axes form a part of the boundary of the safe region. The PartialSafetyCheck algorithm traces the remainder of this boundary.

Figure 3.10 shows the flow of the algorithm while tracing the boundary between the safe (green) and violating (red) regions of the selectivity space for a pair of plans. In this figure, the top and right boundaries of the region violate the safety requirement.

We start from the first violating point on the top-boundary of the grid, and at each stage either move down or right in the grid. At each interior point that we move to, we perform the costing of the plans P_{oe} and P_{re}. The algorithm stops when we reach the bottom or right boundaries of the grid.

The number of *right* or *down* movements required to reach this termination situation is atmost $2r$ movements for a $r \times r$ grid. Hence, for a pair of plans, atmost $4r$ extra costings are needed to obtain the error-resistant area. Steps 1 through 3 of the algorithm require an additional $12r$ costings in the worst case scenario – it is usually much smaller. Thus, the overall time complexity of the modified PartialSafetyCheck algorithm for a 2-dimensional selectivity space is $O(r)$.

The PartialSeer reduction algorithm, which employs the PartialSafetyCheck safety-checking technique, is shown in Figure 3.11.

PartialSafetyCheck (Plan Diagram P, Threshold λ, Area $allowedViolation$, Plan P_{oe}, Plan P_{re}, Dimension d)

1. if(d == 2)
   (a) if (WEDGE_TEST (P,P_{oe},P_{re},λ) == Safe) **return** $allowedViolation$
   (b) if (PERIMETER_TEST (P,P_{oe},P_{re},λ) == Safe) **return** $allowedViolation$
   (c) if the slope criteria of the six conditions of Theorem 3.2 are not satisfied, **return** -1
   (d) if $(allowedViolation = 0)$ **return** -1
   (e) if no *two* adjacent boundaries are safe, **return** -1.
   (f) Let the first violating point at the top-boundary of the grid G occur at $x = x_v$. Set $x = x_v$, $y = r - 1$, $NumViolatingPoints = 0$
   (g) While $x \neq r$ and $y \neq -1$
      i. Set $count = 0$
      ii. While current point is violating (i.e $f(x, y) > 0$) and $y \neq -1$
         A. move down (i.e. y--)
         B. if $(NumViolatingPoints + (r - y - 1) \times (r - x - 1)) > allowedViolation$, **return** -1
      iii. While current point is safe (i.e $f(x, y) \leq 0$) and $x \neq r$
         A. move right (i.e. x++), $count$++
         B. if $(NumViolatingPoints + count \times (r - y - 1)) > allowedViolation$, **return** -1
      iv. $NumViolatingPoints += count \times (r - y - 1)$
   (h) $allowedViolation -= NumViolatingPoints$
   (i) **return** $allowedViolation$
2. else
   (a) for each $(d - 1)$-dimension slice P' of P
      i. $allowedViolation$ = PartialSafetyCheck (P',λ, $allowedViolation$, P_{oe}, P_{re}, $d-1$)
      ii. if $(allowedViolation < 0)$
      return $allowedViolation$;
3. **return** $allowedViolation$.

Figure 3.9: The PartialSafetyCheck Algorithm

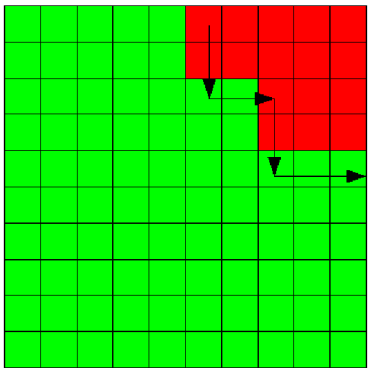


Figure 3.10: Working of the PartialSafetyCheck algorithm

PartialSeer (Plan Diagram P, Threshold λ, MinSafeFraction MSF)

1. Create a Set-Cover Instance $I = (U, S)$, where $S = \{S_1, S_2, ..., S_n\}$, $U = \{1, 2, ..., n\}$, corresponding to the n plans in the original plan diagram P.
2. Set each $S_i = \{i\}, \forall i = 1...n$
3. For each pair of plans (P_i, P_j) do
   (a) Set $allowedViolation = (1 - MSF) \times Area(\mathsf{P})$.
   (b) if (PartialSafetyCheck (P,λ,$allowedViolation$, P_i,P_j,d) ≥ 0) then
   $S_i = S_i \bigcup \{j\}$
4. Solve the set-cover instance I using the Greedy Setcover algorithm to identify the plans retained in R.

Figure 3.11: The PartialSeer Reduction Algorithm

Chapter 4

Experimental Results

The testbed used in our experiments is the Picasso optimizer visualization tool [20], executing on a Sun Ultra 20 workstation equipped with an Opteron Dual Core 2.5GHz processor, 4 GB of main memory and 720 GB of hard disk, running the Windows XP Pro operating system. The experiments were conducted over plan diagrams produced from a variety of multi-dimensional **TPC-H** and **TPC-DS**-based query templates. In our discussion, we use QT*x* to refer to a query template based on Query *x* of the TPC-H benchmark, and DSQT*x* to refer to a query template based on Query *x* of the TPC-DS benchmark. The TPC-H database was of size 1GB, while the TPC-DS database occupies 100GB. We present representative results here for a commercial optimizer anonymously referred to hereafter as OptCom, and a public-domain optimizer, hereafter referred to as OptPub.

4.1 Performance of SIGHT

We now evaluate the two plan generation algorithms, SIGHT and ISIGHT. For this experiment, the OptPub engine was modified to (a) implement the FPC feature internally, and (b) to return the second best sibling plan along with the optimal plan when the “explain” command is executed.

Dimension/ Resolution	Query Template	No. of Plans	Exhaustive Generation time	Time taken by SIGHT	Optimizations performed by SIGHT (%)
2D: 1000 × 1000	QT5	22	5 hrs 20 mins	4 mins (1%)	0.17 %
	QT8	20	6 hrs 10 mins	2 hrs 47 mins (45%)	44 %
3D: 100 × 100 × 100	QT5	23	5 hrs 48 mins	13mins (3%)	2.4 %
	QT8	49	5 hrs 58 mins	2 hrs 2 mins (34%)	32 %
	QT9	22	6 hrs 45 mins	5 mins (1%)	0.24 %
4D: 30 × 30 × 30× 30	QT5	37	4 hrs 50 mins	25 mins (8%)	5.8 %
	QT9	28	6 hrs 10 mins	7 mins (2%)	0.7 %

Table 4.1: Performance of SIGHT algorithm

4.1.1 SIGHT

Using the approximate second best plan, we were able to obtain plan diagrams with no plan identity error, and *almost zero* plan location error. As can be seen in Table 4.1, SIGHT usually requires at most 10% optimizations to generate *close to accurate* plan diagrams for all query templates, except those based on Query 8, the reason for which is discussed below. The good performance of SIGHT can be attributed to the following: Along with the optimizations being performed at select points, all points (except the origin) are costed exactly once. Further, since the FPC feature is internalized in the optimizer, the ratio of plan-costing to plan-searching is approximately 1:100, making the overheads incurred very small. Also, an important byproduct of this minor investment is the ability to also obtain the cost diagram corresponding to the plan diagram.

Though an investment of 10% optimizations is usually the order of the day, there are occasional scenarios when the SIGHT algorithm requires a substantially larger number of optimizations to generate the plan diagram. Such a situation is seen for QT8 – the reason is that the cost of the second best plan is extremely close to that of the optimal plan over an extended region. Even though the actual plan switch occurs much later, this close-to-optimal cost causes the algorithm to optimize at frequent intervals as the constraint $c_1(q') \leq c_2(q)$ is easily violated leading to the algorithm "panicking too quickly" and choosing to optimize a large number of unnecessary points.

Dimension/ Resolution	Query Temp--late	No. of Plans	Exhaustive Generation Time	Approximation Time Taken	Optimizations Required by ISIGHT (%)	ISIGHT Error (%)	
						ϵ_I	ϵ_L
2D: 1000 × 1000	QT5	22	5 hrs 20 mins	3 mins (1%)	0.1 %	13 %	11 %
	QT8	20	6 hrs 10 mins	6 mins (2%)	0.5 %	10 %	11 %
3D: 100 × 100 × 100	QT5	23	5 hrs 48 mins	7 mins (2%)	0.95 %	9 %	4.6 %
	QT8	49	5 hrs 58 mins	12 mins (4%)	1.8 %	16 %	0.2 %
	QT9	22	6 hrs 45 mins	5 mins (2%)	0.4 %	0 %	4.9 %
4D: 30 × 30 × 30× 30	QT5	37	4 hrs 50 mins	15 mins (5%)	3 %	8 %	1 %
	QT9	28	6 hrs 10 mins	7 mins (2%)	0.4 %	3 %	4.5 %

Table 4.2: Performance of ISIGHT algorithm ($\epsilon = 10\%$)

4.1.2 ISIGHT

Turning our attention to the ISIGHT algorithm, whose performance is presented in Table 4.2 for a 10% error bound, we find that it consistently generates approximate plan diagrams while performing less than 5% optimizations. Further and very importantly, even for the problematic QT8, due to the relaxation of the effect of the proximity of the second best plan, the plan diagram is now obtained incurring only a small overhead. Finally, note that the identity errors greater than 10% are usually an artifact of the low number of plans in the original plan diagram.

A related point to note is that the time overheads are a little more than that of optimization. The reason is that, although FPC is very cheap, since it has to be invoked for a very large number of points, a small but perceptible time overhead results.

4.2 Evaluation of SEER

4.2.1 Experimental Setup

Physical Design. Following a methodology similar to that advocated in [6], we considered three different physical design configurations in our study: **PrimaryKey (PK)**, **AllIndex (AI)**, and **TunedIndex (TI)**. PK represents the default physical design of our database engine, wherein a clustered index is created on each primary key. AI, on the other hand, represents an "index-rich" situation wherein (single-column) indices are available on all query-related schema attributes. Finally, TI represents the index environment obtained by implementing the recommendations of the database engine's index tuning advisor (which include multi-column

indices).

In the subsequent discussion, we use QT*x* to refer to a query template based on Query *x* of the TPC-H benchmark, and DSQT*x* to refer to a query template based on Query *x* of the TPC-DS benchmark, operating in the default PK configuration. We prefix AI and TI to the query template identifiers in describing our results for these specialized configurations.

Query Location Distribution. All the performance results shown initially in this section are for plan diagrams generated with *exponentially* distributed locations for the query points across the selectivity space, resulting in higher query densities near the selectivity axes and towards the origin. This choice is based on earlier observations in the literature (e.g. [12, 13, 21]) that plans tend to be densely packed in precisely these regions of the selectivity space. From a performance perspective, these diagrams represent the "tough-nut" challenging situations with respect to obtaining anorexic reduction due to their high plan densities and substantially broader range of plan cost values.

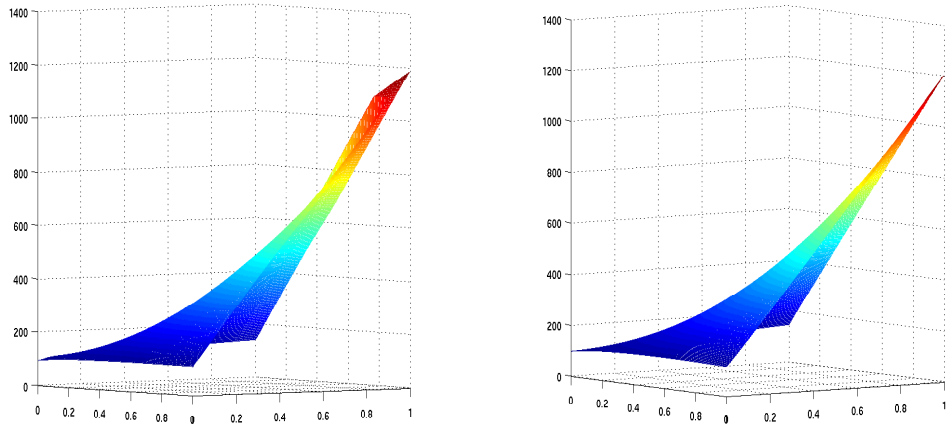
For completeness, we have also conducted all the experiments with a *uniform* distribution of query locations – these results are detailed in Section 4.2.7.

Performance Metrics. In the remainder of this section, we evaluate the SEER reduction algorithm with regard to the following performance parameters: (a) Diagram Reduction Quality, (b) Error-resistance obtained through Reduction, (c) Safety of Reduction, and (d) Computational Efficiency. As a precursor, we first evaluate the validity of the plan cost function model (Section 3.2.1).

4.2.2 Validity of Plan Cost Model

The validity of the plan cost model presented in Equation 3.1 was assessed by attempting to fit the costs of plans generated by OptCom[1]. The experimental data consisted of optimizer-estimated execution costs over the selectivity space of the plans that appeared in the various plan diagrams (taken from both exponentially and uniformly distributed query templates). As

[1]We have also validated this plan cost model on another commercial database, and found the results to be similar



(a) **Actual Cost Function** (b) **Fitted Cost Function**

Figure 4.1: Plan Cost Function Modeling

mentioned earlier, the foreign-plan-costing (FPC) feature was used to evaluate plans outside of their endo-optimal regions.

The surface fitting was carried out with the classical Linear Least Squares method [16] and implemented using Matlab 7.4 [25]. An example 2-D fitted cost function is:

$$Cost(x, y) = 17.9x + 45.9y + 1046xy - 39.5x \log x + 4.5y \log y + 27.6xy \log xy + 97.3$$

For this plan, the complete plots of the actual cost surface and the fitted cost surface, as a function of the selectivities of the two base relations, are shown in Figure 4.1. It is visually evident that the fit is very good.

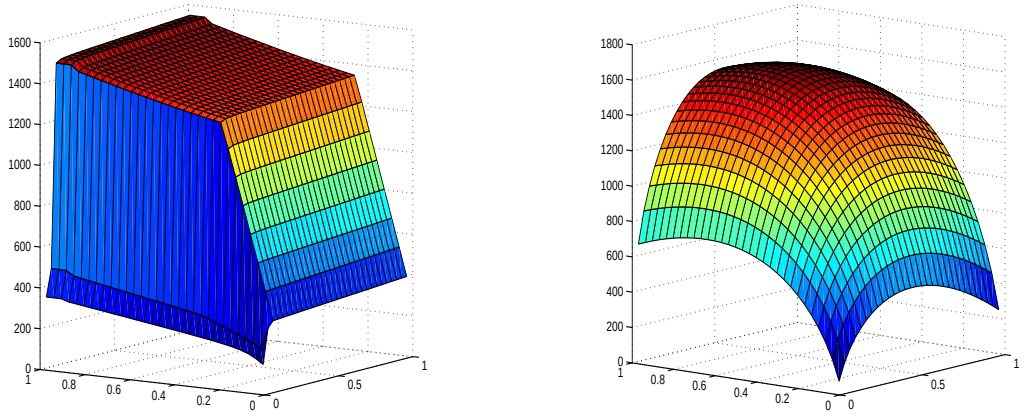
As further evidence of the accuracy of our model, Table 4.3 shows the quality-of-fit, measured in terms of the maximum and average *Root-Mean-Square(RMS)* errors, over a large number of plans featuring in the plan diagrams arising from our suite of multi-dimensional query templates. The consistently low RMS values suggest that the model is sufficiently accurate for our purposes.

Finally, as an additional precaution, we deliberately searched for plan cost functions with complex shapes to assess the quality-of-fit in these difficult cases. An example is shown in Figure 4.2, and we see that even here, the fit is of high quality (the RMS Error is only around 10%). This can be attributed to the fact that our cost model has 7 parameters which gives

Dimension	Number of Plans	Maximum RMS Error (%)	Average RMS Error(%)
2D (TPC-H)	614	14.20	1.82
2D (TPC-DS)	168	7.31	2.87
3D (TPC-H)	28	6.98	1.92
3D (TPC-DS)	100	2.71	1.58

Table 4.3: RMS Errors in Fitted Cost Surfaces

sufficient freedom to fit most of the plan cost functions found in practice. Our curve fitting technique does not impose any restriction on the behaviour of the cost function, and hence we see for this example, a small PCM violation in the fitted curve.



(a) Actual Cost Function (b) Fitted Cost Function

$$Cost(x,y) = 184.3x + 619.9y + 524.5xy - 1090x\log x - 1179.9y\log y - 836.2xy\log xy - 1000$$

$$RMSError = 10.96\%$$

Figure 4.2: Complex Plan Cost Function

4.2.3 Plan Diagram Reduction Quality

A potentially worrisome aspect of our quest to obtain globally robust reduction is whether it might result in losing out on the anorexic reduction levels observed in the localized reduction processes of [11]. This concern is quantitatively allayed in Table 4.4, which presents a comparison between SEER and CostGreedy (CG) of the number of plans in the reduced diagram for

a diverse suite of multi-dimensional query templates on the TPC-H database. The PK physical design configuration was operational in these experiments.

At first glance, SEER might have been expected to perform worse than CostGreedy because its additional safety checks may prevent some plan swallowings permitted by CostGreedy– in fact, this was the source of our concern. However, in Table 4.4, we actually find the *converse* – while CostGreedy does provide anorexic reduction, SEER does even better. The reason for this is that CostGreedy follows a conservative cost-bounding approach to estimate the costs of plans outside their endo-optimal regions (details in [11]). SEER, on the other hand, uses the foreign-plan-costing feature to obtain the exact costs in these regions, and therefore has superior reduction possibilities. Therefore, the FPC feature comes in handy from both quality and safety perspectives.

A question that immediately arises is how SEER would compare against a CostGreedy variant that also utilized the FPC feature. This issue is also addressed in Table 4.4, where the performance of this variant (CG-FPC) is presented. We see that CG-FPC does perform better or as well as SEER, as should be expected – however, the gap, if any, is always very small. A related point to note here is that the SEER reduction quality remains excellent even for the 3D query templates, in spite of the fact that the additional dimension increases the possibility of the safety conditions being violated.

Finally, we observe in Table 4.4 that the LiteSEER fast variant happens to provide reduction quality identical to SEER. Under the AI (and TI) configurations, however, it occasionally performs slightly better (see Section 4.2.3), as should be expected due to its being less stringent in allowing replacements.

TPC-DS Results. The above results were generated on the TPC-H database, which has uniformly distributed data. Table 4.5 shows a corresponding set of results for plan diagrams generated on the TPC-DS database, which features skewed data. It is immediately evident that the reduction profiles of the various reduction algorithms are very similar to those seen with TPC-H.

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
QT2 (2D)	60	14	3	6	6
QT5 (2D)	51	7	2	2	2
QT8 (2D)	121	7	2	2	2
QT9 (2D)	137	9	3	4	4
QT10 (2D)	44	3	3	3	3
QT16 (2D)	32	11	3	3	3
QT5 (3D)	68	8	3	3	3
QT8 (3D)	191	8	3	3	3
QT10 (3D)	75	10	3	4	4

Table 4.4: Plan Diagram Reduction Quality (TPC-H)

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
DSQT12 (2D)	25	6	3	2	2
DSQT18 (2D)	114	13	2	2	2
DSQT19 (2D)	55	11	3	4	4
DSQT12 (3D)	33	11	2	2	2
DSQT18 (3D)	222	15	2	4	4
DSQT19 (3D)	98	15	2	4	4

Table 4.5: Plan Diagram Reduction Quality (TPC-DS)

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
AIQT2 (2D)	87	12	2	2	2
AIQT5 (2D)	126	14	4	6	5
AIQT8 (2D)	121	7	3	3	3
AIQT9 (2D)	132	14	3	4	3
AIQT10 (2D)	37	8	4	5	5
AIQT16 (2D)	35	9	2	2	2
AIQT5 (3D)	139	14	5	7	5
AIQT8 (3D)	168	14	4	6	5
AIQT10 (3D)	77	16	7	8	8

Table 4.6: Plan Diagram Reduction Quality (TPCH-AI)

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
TIQT2 (2D)	52	10	4	5	5
TIQT8 (2D)	108	16	3	3	3
TIQT9 (2D)	101	16	6	5	5
TIQT10 (2D)	50	10	4	2	2
TIQT16 (2D)	36	14	4	6	6
TIQT5 (3D)	84	10	4	5	5
TIQT8 (3D)	181	14	4	6	5
TIQT10 (3D)	78	12	6	8	7

Table 4.7: Plan Diagram Reduction Quality (TPCH-TI)

Reduction Quality with AllIndex Configuration

While the PK configuration had only 8 primary-key indices, AllIndex includes an additional 53 (non-clustered) single-column indices covering all the remaining query-related schema attributes. The reduction quality results for this index-rich configuration are shown in Table 4.6. We first notice that the number of plans in the original diagram usually increases, often substantially, as should be expected since the optimizer's search space has increased due to the availability of the additional indices. For example, the number of plans for AIQT5(2D) goes up to 125 from 51, while AIQT5(3D) jumps to 139 from 68. However, when we consider the reduction quality of the various algorithms, we find that they continue to *materially adhere to anorexic levels*, although the actual cardinalities may have gone up by a couple of plans. For example, SEER on AIQT5(2D) retains 6 plans as compared to 2 under PK.

Another point to note in Table 4.6 is that we now see LiteSEER occasionally permitting slightly greater reduction than SEER, due to its relaxed constraint in allowing replacements.

Reduction Quality with Tuned-Index Configuration

The reduction quality results for the *Tuned Index* (TI) configuration which implements the recommendations of the index tuning advisor shipped with OptCom is shown in Table 4.7. The parameters of the tuning advisor were set to their default values, and the TPC-H benchmark queries (generated with the QGen utility) formed the input workload. For this setup, the advisor recommended 20 additional indices beyond the default Primary Key configuration.

We see here that the reduction performance is very similar to that obtained with the PK and AI configurations, testifying to SEER's consistent behavior over a wide variety of database environments.

4.2.4 Error-resistance and Safety

Having established the retention of diagram reduction quality, we now move on to assessing the extent to which resistance to selectivity errors is provided through SEER reduction. We begin with defining a metric that quantitatively captures this effect:

Error Resistance Metric. Given an estimated query location q_e and an actual location q_a, the *Selectivity Error Resistance Factor* **(SERF)** of a replacement plan P_{re} w.r.t. the optimal plan P_{oe} is defined as,

$$SERF(q_e, q_a) = 1 - \frac{c_{re}(q_a) - c_{oa}(q_a)}{(1+\lambda)c_{oe}(q_a) - c_{oa}(q_a)}$$

Intuitively, SERF captures the fraction of the performance gap between P_{oe} and P_{oa} that is closed by P_{re}. In principle, SERF values can range over $(-\infty, 1]$, with the following interpretations: SERF in the range $(\lambda, 1]$ indicates that the replacement is beneficial, with values close to 1 implying "immunity" to the selectivity error. For SERF in the range $[0, \lambda]$, the replacement is indifferent in that it neither helps nor hurts, while SERF values below 0 highlight a harmful replacement that materially worsens the performance.

The above formula applies to a specific instance of replacement. To capture the net impact of reduction on improving the resistance in an *entire plan diagram*, we compute the following

$$AvgSERF = \frac{\sum_{q_e \in rep(\mathsf{P})} \sum_{q_a \in exo_{oe}(\mathsf{P})} SERF(q_e, q_a)}{\sum_{q_e \in rep(\mathsf{P})} \sum_{q_a \in exo_{oe}(\mathsf{P})} 1}$$

where $rep(\mathsf{P})$ is the set of points in the plan diagram P that were replaced during the reduction process, and $exo_{oe}(\mathsf{P})$ is the set of points lying in the exo-optimal region defined with respect to P_{oe}, the optimizer's plan choice for q_e. The normalization is with respect to the number of possible selectivity errors in the diagram. (To ensure meaningful AvgSERF values from a robustness perspective, we exclude the uninteresting scenarios wherein both c_{re} and c_{oe} have

extremely low absolute values, or are both within λ-threshold of c_{oa}.)

Note that in the above formulation, we assume for simplicity that the actual location q_a is equally likely to be anywhere in P_{oe}'s exo-optimal space, that is, that the errors are uniformly distributed over this space. However, our conceptual framework is also applicable to the more generic case where the error locations have an associated probability distribution.

Resistance Results. For CostGreedy, SEER and LiteSEER, we show in Table 4.8, the AvgSERF, as defined above, as well as MinSERF and MaxSERF, the minimum and maximum values of SERF over all replacement instances, for the various query templates. We first see here that for all the algorithms, plan diagram reduction is capable, across the board, of providing complete immunity (MaxSERF tending to 1) to selectivity errors for individual replacement instances. Secondly, and more importantly, the AvgSERF is also quite substantial for SEER. For example, in DSQT18, on average, more than three-quarters of the performance gap due to selectivity errors is bridged by the SEER reduction process.

With CostGreedy, on the other hand, the AvgSERF is comparatively very poor, and occasionally even negative! The important point to note here is that these low averages are an artifact arising out of a small fraction of points (around 10-20% points occuring with probability of around 0.1) whose performance is grossly adversely affected by plan replacement. That is, plan reduction does help in the vast majority of cases but the "few very bad apples", reflected by the hugely negative MinSERF values (which sometimes even run into the thousands), ruin the overall performance statistics. More pertinently, these results serve to quantitatively and vividly substantiate the need for safe replacement, the motivation underlying our design of the SEER algorithm.

Finally, turning our attention to LiteSEER, we see that its error-resistance profile is very similar to that of SEER – in fact, the AvgSERF and MaxSERF numbers are identical for most templates. Further, although like CostGreedy it does not guarantee safety, as testified to by the negative values in the MinSERF column, note that (a) the templates having negative values are relatively rare, (b) even in these cases, unsafe replacements occur only for about 1% of the points (with probability less than 0.01), and (c) most importantly, their magnitudes are small in comparison (the maximum is -10 for AIQT5(2D)).

Query Template	CG			SEER			LiteSEER		
	MinSERF	**AvgSERF**	**MaxSERF**	**MinSERF**	**AvgSERF**	**MaxSERF**	**MinSERF**	**AvgSERF**	**MaxSERF**
QT2 (2D)	-58.6	0.2	1	0	0.38	0.98	0	0.38	0.98
QT5 (2D)	-15.6	0.5	1	0.14	0.51	0.99	0.14	0.51	0.99
QT8 (2D)	-2.9	0.82	1	0.3	0.93	1	0.3	0.93	1
QT9 (2D)	-46.3	0.37	1	0	0.77	1	0	0.77	1
QT10 (2D)	-23.8	0.09	1	0	0.35	1	0	0.35	1
QT16 (2D)	-132.5	0.03	0.99	0.04	0.33	0.96	0.04	0.33	0.96
QT5 (3D)	-15.4	0.6	1	0.03	0.38	1	0.03	0.38	1
QT8 (3D)	-177.7	-0.5	1	0	0.63	1	0	0.63	1
QT10 (3D)	-130.7	-2	1	0	0.45	1	0	0.45	1
AIQT2 (2D)	-202	-1.1	0.99	0	0.8	0.99	0	0.8	0.99
AIQT5 (2D)	-1336	-3.1	1	0	0.58	1	-10	0.54	1
AIQT8 (2D)	-62.4	0.35	1	0	0.54	1	0	0.54	1
AIQT9 (2D)	-9486	-3.1	1	0	0.62	1	-5	0.66	1
AIQT10 (2D)	-20.2	0.24	1	0	0.25	0.98	0	0.25	0.98
AIQT16 (2D)	-76	0.07	1	0	0.75	1	0	0.75	1
AIQT5 (3D)	-2151	0.24	1	0.05	0.62	1	-2	0.66	1
AIQT8 (3D)	-103.7	0.43	1	0	0.44	1	-6	0.44	1
AIQT10 (3D)	-4680	-4.4	1	0	0.33	1	0	0.33	1
TIQT2 (2D)	-161.3	-0.12	0.99	0.01	0.32	0.94	0.01	0.32	0.94
TIQT8 (2D)	-550.1	-3.4	1	0	0.54	1	0	0.54	1
TIQT9 (2D)	-87.5	0.01	1	0	0.44	1	0	0.44	1
TIQT10 (2D)	-671.4	-15.7	1	0	0.27	0.99	0	0.27	0.99
TIQT16 (2D)	-14.4	0.11	0.99	0.02	0.39	0.97	0.02	0.39	0.97
TIQT5 (3D)	-166.3	-1.42	1	0	0.71	1	0	0.71	1
TIQT8 (3D)	-4188.5	-4.23	1	0	0.61	1	-1.3	0.61	1
TIQT10 (3D)	-574	-4.97	1	0	0.31	1	-8.9	0.31	1
DSQT12 (2D)	-4	0.61	1	0.07	0.44	1	0.06	0.44	1
DSQT18 (2D)	-194.75	-2	1	0.07	0.81	1	0.07	0.81	1
DSQT19 (2D)	-86	0.17	1	0	0.63	1	0	0.63	1
DSQT12 (3D)	-142	-0.32	1	0.06	0.53	1	0.06	0.53	1
DSQT18 (3D)	-3104	-0.14	1	0	0.85	1	0	0.85	1
DSQT19 (3D)	-106.6	0.33	1	0.02	0.82	1	0.02	0.82	1

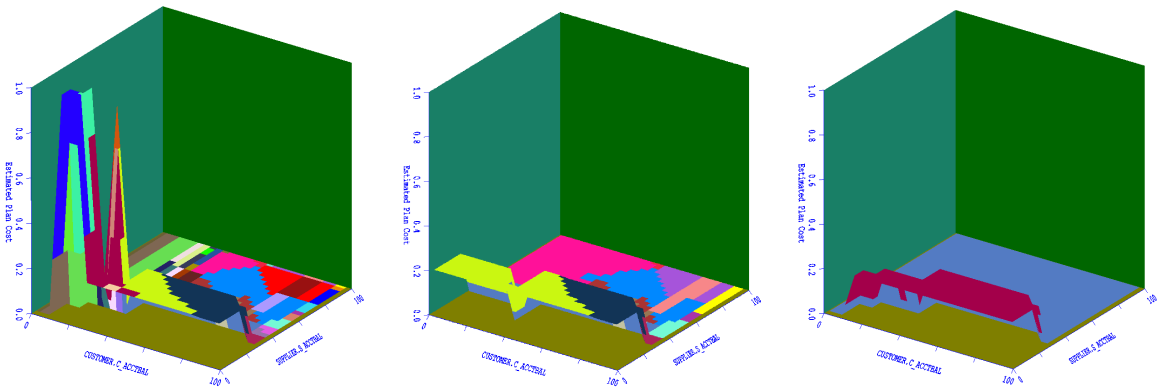
Table 4.8: Characterization of Error-Resistance through Reduction

Safety Example

In the example of Figure 3.2, plan diagram reduction without explicitly checking for safety led to situations wherein P_{re} performed much worse than P_{oe} at q_a. The effectiveness of SEER in avoiding such unsafe replacements is visually highlighted in the sequence of pictures in Figure 4.3, corresponding to the same example.

Assuming that the actual location of a query at run-time q_a is uniformly distributed over S, Figure 4.3(a) shows the *expected cost* for each query point q_e, when executed with its optimizer-selected plan P_{oe}. Note that the peaks in the picture correspond to situations where the plan-choice is highly sensitive to selectivity errors.

Then, Figure 4.3(b) shows the expected cost of each query point q_e when executed with P_{re} from the reduced plan diagram obtained using CostGreedy. Note that virtually all the peaks in Figure 4.3(a) are substantively eliminated through the replacement choices in the reduced



(a) Expected Costs (Optimizer) (b) Expected Costs (CG) (c) Expected Costs (SEER)

Figure 4.3: Safe Error-resistance with SEER

plan diagram – for example, the dark-blue peak at the left-top corner of Figure 4.3(a) is largely removed. However, on the down side, some plans suffer injurious replacements – for e.g., the earth-brown colored plan in the left-bottom corner of Figure 4.3(a) is now replaced by the fluorescent-green colored plan in Figure 4.3(b), whose expected cost is orders of magnitude greater. That is, CostGreedy in the process of eliminating existing peaks, may introduce *new peaks*.

Finally, in Figure 4.3(c), we show the performance of SEER reduction. We see here that (a) it removes all the peaks of Figure 4.3(a) like CostGreedy, and (b) it does not introduce any new peaks courtesy its safety criterion. In a nutshell, "it provides virtually all the good, and doesn't introduce any harm".

4.2.5 Efficiency of Reduction Process

We now move on to profiling the time taken to complete the reduction process by SEER as compared to CostGreedy. These results are shown in Table 4.9 for our query template suite.

Focussing initially on the 2D query templates, we see that SEER's performance is quite acceptable in terms of absolute times (a few minutes per reduction), especially in comparison to the original plan diagram *production time*. However, it is much slower relative to CostGreedy, which offers sub-second response times. This might seem surprising in light of our analysis in

Query Template	CG (ms)	CG-FPC (min)	SEER (min)	LiteSEER (sec)
QT2 (2D)	15	53.1	3.6	14.2
QT5 (2D)	16	45.0	1.0	12
QT8 (2D)	17	108	9.6	28.8
QT9 (2D)	13	122.4	10.6	32.6
QT10 (2D)	15	38.7	3.0	10.3
QT16 (2D)	15	27.9	1.3	7.5
QT5 (3D)	25	67	19.0	32
QT8 (3D)	21	190	65.0	91
QT10 (3D)	17	74	16.5	4.5
DSQT12 (2D)	14	21.6	2.6	5.8
DSQT18 (2D)	13	101.7	9.4	27.1
DSQT19 (2D)	14	48.6	6.4	13.0
DSQT12 (3D)	20	32.0	7.4	15.4
DSQT18 (3D)	25	221.0	89.1	106.1
DSQT19 (3D)	23	97.0	35.8	46.6

Query Template	CG (ms)	CG-FPC (min)	SEER (min)	LiteSEER (sec)
AIQT2 (2D)	17	77.4	5.0	20.6
AIQT5 (2D)	12	112.5	3.7	30.0
AIQT8 (2D)	11	108.0	6.9	28.8
AIQT9 (2D)	18	107.9	9.1	31.4
AIQT10 (2D)	12	32.4	2.0	8.6
AIQT16 (2D)	12	30.6	2.0	8.2
AIQT5 (3D)	26	138	37.7	66.2
AIQT8 (3D)	19	167	47.3	80.2
AIQT10 (3D)	24	76	14.9	36.5
TIQT2 (2D)	18	45.9	2.9	12.2
TIQT8 (2D)	12	96.3	4.9	25.7
TIQT9 (2D)	16	90.0	7.2	24.0
TIQT10 (2D)	14	44.1	2.6	11.8
TIQT16 (2D)	12	31.5	2.0	8.4
TIQT5 (3D)	28	83	20.8	39.8
TIQT8 (3D)	24	180	67.8	86.4
TIQT10 (3D)	19	78	15.9	37.0

Table 4.9: Efficiency of Reduction Process

Section 3.3 showing that SEER is an $O(n\sqrt{m}+n^2)$ algorithm, whereas CostGreedy is $O(nm)$. The reason for the higher running time of SEER is that the basic cost-bounding computation in CostGreedy is much faster than the foreign-plan-costing operator provided by the commercial optimizers. Our discussions with the development team of OptCom have indicated that this is not due to the costing itself, but is largely an artifact of setting up the contexts for the costing, including verifying the validity of the plan with respect to the query. Therefore, it is possible that future better implementations of the FPC feature may bring SEER's running time closer to CostGreedy.

When we consider the 3D query templates, however, the running times of SEER can be quite large. It is here that LiteSEER shows its worth since its running times are only a few minutes or even less, across the board for all the query templates. Taken in conjunction with its good safety performance (Section 4.2.4), it suggests that LiteSEER offers an extremely attractive compromise between the speed of CostGreedy and the robustness of SEER, making it a viable first-cut reduction technique in real-world installations.

Finally, to normalize the effect of the different costing implementations, the running time of the CG-FPC algorithm is also shown in Table 4.9 – we see here that CG-FPC takes in the order of *several tens or few hundreds of minutes* to complete the reduction process. In comparison, SEER's selective usage of the FPC operator, courtesy Theorem 3.2 and the two-stage checking process, does succeed in substantially bringing down the overheads.

4.2.6 Performance of PartialSeer

Sample results obtained with the PartialSeer algorithm for $MSA = 0.8$ are shown in Table 4.10, 4.11 and 4.12, for the various metrics of reduction quality, error resistance and efficiency of the reduction.

Query Template	Original No. of plans	CG	CG-FPC	SEER (MSA = 1)	PartialSeer (MSA = 0.8)
QT4(2D)	16	7	3	3	2
QT5(2D)	51	10	2	2	1
QT16(2D)	32	11	3	3	3
DSQT18(2D)	114	13	2	2	2

Table 4.10: Plan Diagram Reduction Quality ($\lambda = 20\%$)

Query	MinSERF	AvgSERF	MaxSERF
QT4(2D)	-0.28	0.36	0.998
QT5(2D)	-15.6	0.37	1
QT16(2D)	-0.25	0.37	0.99
DSQT18(2D)	-0.26	0.83	1

Table 4.11: Error-Resistance of PartialSeer ($\lambda = 20\%, MSA = 0.8$)

Query	No of FPC	Time (min)
QT4(2D)	2780	2.8
QT5(2D)	8738	8.7
QT16(2D)	5046	5.0
DSQT18(2D)	27284	27.3

Table 4.12: Efficiency of PartialSeer ($\lambda = 20\%, MSA = 0.8$)

4.2.7 Uniform Query Distribution

The results shown thus far were produced with an exponential distribution of query points across the selectivity space. We present here the corresponding results for plan diagrams generated with a *uniform* distribution of query points. Tables 4.13 and 4.14 show the reduction quality over our suite of query templates on the TPC-H and TPC-DS databases, respectively, operating with a PrimaryKey physical configuration. The performance on the AllIndex and TunedIndex configurations are detailed in Tables 4.15 and 4.16, respectively. Finally, the error-resistance quality and the reduction efficiency are shown in Tables 4.17 and 4.18, respectively.

These results are behaviorally similar to those obtained with the exponential distribution.

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
QT2 (2D)	25	5	3	3	3
QT5 (2D)	10	3	1	1	1
QT8 (2D)	31	4	2	2	2
QT9 (2D)	21	2	1	1	1
QT10 (2D)	13	3	2	2	2
QT16 (2D)	26	9	2	3	3
QT5 (3D)	18	1	1	1	1
QT8 (3D)	18	6	3	3	3
QT10 (3D)	18	4	2	2	2

Table 4.13: Plan Diagram Reduction Quality (TPC-H, Uniform plan diagrams)

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
DSQT12 (2D)	7	4	2	2	2
DSQT18 (2D)	21	3	1	1	1
DSQT19 (2D)	28	5	2	2	2
DSQT12 (2D)	8	2	1	1	1
DSQT18 (3D)	36	2	1	1	1
DSQT19 (2D)	64	2	1	1	1

Table 4.14: Plan Diagram Reduction Quality (TPC-DS, Uniform plan diagrams)

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
AIQT2 (2D)	30	8	3	3	3
AIQT5 (2D)	25	6	2	2	2
AIQT8 (2D)	25	3	2	3	3
AIQT9 (2D)	25	5	1	1	1
AIQT10 (2D)	16	4	3	3	3
AIQT16 (2D)	22	14	3	4	4
AIQT5 (3D)	37	4	2	2	2
AIQT8 (3D)	39	5	2	3	3
AIQT10 (3D)	50	9	4	3	3

Table 4.15: Plan Diagram Reduction Quality (TPCH-AI, Uniform plan diagrams)

Query Template	Original No. of plans	CG	CG-FPC	SEER	LiteSEER
TIQT2 (2D)	25	5	3	4	4
TIQT8 (2D)	29	2	1	1	1
TIQT9 (2D)	49	12	3	5	3
TIQT10 (2D)	8	3	2	2	2
TIQT16 (2D)	20	10	3	3	3
TIQT5 (3D)	30	5	3	3	3
TIQT8 (3D)	35	6	2	2	2
TIQT10 (3D)	16	6	2	2	2

Table 4.16: Plan Diagram Reduction Quality (TPCH-TI, Uniform plan diagrams)

Query Template	CG			SEER			LiteSEER		
	MinSERF	**AvgSERF**	**MaxSERF**	**MinSERF**	**AvgSERF**	**MaxSERF**	**MinSERF**	**AvgSERF**	**MaxSERF**
QT2 (2D)	0	0.76	0.98	0.02	0.62	0.98	0.02	0.62	0.98
QT5 (2D)	0.99	0.99	0.99	0.96	0.98	1	0.96	0.98	1
QT8 (2D)	-0.47	0.73	0.99	0.27	0.44	0.99	0.27	0.44	0.99
QT9 (2D)	-55.3	-0.56	1	0.86	0.99	1	0.86	0.99	1
QT10 (2D)	-0.81	0.4	0.99	0.14	0.34	0.60	0.14	0.34	0.60
QT16 (2D)	-0.33	0.49	0.97	0.04	0.46	0.92	0.04	0.46	0.92
QT5 (3D)	0	0.01	0.02	0.78	0.95	1	0.78	0.95	1
QT8 (3D)	-22.6	0.41	1	0.15	0.36	0.99	0.15	0.36	0.99
QT10 (3D)	-5.5	0.43	1	0.19	0.69	1	0.19	0.69	1
AIQT2 (2D)	-3.95	0.16	0.97	0.15	0.64	0.94	0.15	0.64	0.94
AIQT5 (2D)	0.17	0.38	1	0.28	0.51	1	0.28	0.51	1
AIQT8 (2D)	-0.27	0.48	0.99	0.19	0.46	0.99	0.19	0.46	0.99
AIQT9 (2D)	-1999.8	-2.1	1	0.96	0.99	1	0.96	0.99	1
AIQT10 (2D)	-5.7	0.32	1	0.16	0.33	0.68	0.16	0.33	0.68
AIQT16 (2D)	-0.39	0.58	0.96	0	0.58	0.98	0	0.58	0.98
AIQT5 (3D)	0.3	0.6	1	0.05	0.9	1	0.05	0.9	1
AIQT8 (3D)	-9.25	0.32	0.99	0.03	0.75	0.99	0.03	0.75	0.99
AIQT10 (3D)	-24.6	0.44	1	0.08	0.5	1	0.08	0.5	1
TIQT2 (2D)	-4.23	0.52	0.98	0.01	0.49	0.95	0.01	0.49	0.95
TIQT8 (2D)	-26.9	0.48	0.92	0.6	0.93	0.99	0.6	0.93	0.99
TIQT9 (2D)	-37.5	0.21	0.99	0	0.32	0.99	-0.27	0.36	0.99
TIQT10 (2D)	0.17	0.18	0.53	0.17	0.32	0.59	0.17	0.32	0.59
TIQT16 (2D)	-4.2	0.52	0.96	0.07	0.56	0.98	0.07	0.56	0.98
TIQT5 (3D)	-96.9	0.43	0.99	0.22	0.89	0.99	0.22	0.89	0.99
TIQT8 (3D)	-9.47	0.22	0.99	0.04	0.67	0.99	0.04	0.67	0.99
TIQT10 (3D)	-2.96	0.52	0.99	0.38	0.9	0.99	0.38	0.9	0.99
DSQT12 (2D)	-1	0.16	1	0.25	0.32	0.54	0.25	0.32	0.54
DSQT18 (2D)	-12.2	0.38	1	0.59	0.99	1	0.59	0.99	1
DSQT19 (2D)	-11.8	0.08	1	0.18	0.47	1	0.18	0.47	1
DSQT12 (3D)	1	1	1	1	1	1	1	1	1
DSQT18 (3D)	0.99	0.99	0.99	0.72	0.86	1	0.72	0.86	1
DSQT19 (3D)	-0.53	0.74	1	0.77	0.87	1	0.77	0.87	1

Table 4.17: Characterization of Error-Resistance through Reduction (Uniform plan diagrams)

Query Template	CG (ms)	CG-FPC (min)	SEER (min)	LiteSEER (sec)
QT2 (2D)	15	21.6	2.2	5.8
QT5 (2D)	14	8.1	0.7	2.2
QT8 (2D)	14	27.0	1.9	7.2
QT9 (2D)	13	18.0	2.1	4.8
QT10 (2D)	14	10.8	0.7	2.9
QT16 (2D)	13	22.5	1.4	6.0
QT5 (3D)	25	17.0	5.1	8.2
QT8 (3D)	21	29.0	10.7	13.9
QT10 (3D)	22	23.0	7.8	11.0
DSQT12 (2D)	19	5.4	0.07	1.4
DSQT18 (2D)	17	18.0	1.2	4.8
DSQT19 (2D)	14	24.3	1.7	6.5
DSQT12 (3D)	20	7.0	1.2	3.4
DSQT18 (3D)	30	35.0	7.2	16.8
DSQT19 (3D)	26	63.0	12.7	30.2

Query Template	CG (ms)	CG-FPC (min)	SEER (min)	LiteSEER (sec)
AIQT2 (2D)	16	26.1	2.8	7.0
AIQT5 (2D)	16	21.6	0.7	5.8
AIQT8 (2D)	14	21.5	2.1	5.8
AIQT9 (2D)	15	21.6	1.7	5.7
AIQT10 (2D)	13	13.5	0.7	3.6
AIQT16 (2D)	13	18.9	0.4	5.0
AIQT5 (3D)	23	36.0	12.0	17.3
AIQT8 (3D)	20	38.0	14.0	18.2
AIQT10 (3D)	20	49.0	14.0	23.5
TIQT2 (2D)	16	21.6	2.3	5.8
TIQT8 (2D)	16	25.2	1.1	6.7
TIQT9 (2D)	16	43.2	2.2	11.5
TIQT10 (2D)	16	6.3	0.43	1.7
TIQT16 (2D)	15	17.1	0.52	4.6
TIQT5 (3D)	16	29	8.7	13.9
TIQT8 (3D)	16	34	11	16.3
TIQT10 (3D)	16	15	2.2	7.2

Table 4.18: Efficiency of Reduction Process (Uniform plan diagrams)

Chapter 5

Conclusions

We have investigated in this thesis, methods for the efficient generation of plan diagrams, a key resource in the analysis and redesign of modern database query optimizers. For optimizers that support *plan rank list* and *foreign plan costing* features, we proved that the SIGHT algorithm produced zero errors and was able to do so incurring overheads of less than 10%. However, it performs poorly for query templates that have the second-best plan being very close to the optimal choice over an extended region. We then demonstrated that the ISIGHT algorithm, which traded error for performance, was able to satisfy the 10% error bound with less than 5% optimizations. It was also able to adequately handle the problem templates of SIGHT.

As a second complementary problem, we investigated whether the optimizer's choices could be replaced by alternative plans from the parametric optimal set over the selectivity space that are more resilient to selectivity estimation errors, which are well-documented causes of poor plan choices by database optimizers. In particular, the recently proposed notion of anorexic reduction of plan diagrams was used to provide replacements that had large endo-optimal regions, making them error-resistant by definition in these areas. Further, the empirical evidence suggested that error-resistance was provided even in the exo-optimal regions in the vast majority of the cases. However, there were occasional situations where the replacement could turn out to be significantly worse. To prevent this, we developed a simple but accurate model of plan cost behavior. To our knowledge, this model is the first such characterization for industrial-strength query optimizers. Using this formulation, we devised efficient checks that

operate only on the boundaries of the space to decide safety in the entire space. The checks are implemented utilizing foreign-plan costing feature provided by the commercial database engines. A particularly attractive feature of our approach is that the safety guarantee applies on an *individual query basis*. As a bonus, the foreign-plan costing, in addition to providing safety, was leveraged to further improve the quality and complexity of the plan diagram reduction process.

The above techniques were integrated into the SEER algorithm and the intended benefits validated on a representative range of TPC-H and TPC-DS-based query templates on leading commercial optimizers. We observed that typically at least one-third of the performance gap due to selectivity errors was bridged by the SEER reduction process, while in some instances, virtually *complete immunity* against selectivity errors was obtained. We also presented Lite-SEER, a light-weight version of SEER that very cheaply provides a high degree of safety by restricting its attention to only the corners of the selectivity space.

In summary, we present SIGHT to efficiently generate high-dimension and high-resolution plan diagrams, with typical overheads being an *order of magnitude lower* than the brute-force approach. We then provide SEER, an effective and safe compile-time mechanism for substantially increasing resistance to selectivity errors on base relations, without requiring modifications to the optimizer or specialized knowledge of the workload/system.

Currently, SEER operates as a post-processor after production of the plan diagram. In future, we intend to investigate how optimizers could internalize these ideas and be redesigned to directly produce safe reduced plan diagrams. Also, while we assumed a uniform distribution of selectivity estimation errors, it would be interesting to extend our results to incorporate user-defined probability distributions.

We hope that our results will encourage all database vendors to incorporate the plan-rank-list and foreign-plan-costing features, both of which were critical to the excellent performance of SIGHT and SEER, in their optimizer APIs.

Appendix A

Plan Diagram Reduction Variants

A.1 Quality of Plan Diagram Reduction

The CostGreedy algorithm given in [11] computes the reduced plan diagram with k plans, such that the cost of every point in the reduced plan diagram is within λ of its original cost. It is possible that there might exist a reduced plan diagram with another set of k plans, where the maximum cost increase is much lower than λ. We would ideally like to obtain the reduced plan diagram with minimum number of plans that also minimizes the maximum cost-increase of the query points in the reduced plan diagram.

To solve this problem, we provide a 2-step approximation algorithm ReduceGreedy, that provides a bounded performance guarantee.

A.1.1 The ReduceGreedy Algorithm

The ReduceGreedy algorithm is given in Figure A.1. In the first step of this algorithm we use CostGreedy to obtain a reduced plan diagram with k plans. We then use the ThresholdGreedy algorithm [11] with a budget of k plans to obtain a reduced plan diagram which minimizes the maximum cost-increase over all query points in the plan diagram.

We know that the the ThresholdGreedy algorithm guarantees that the solution is atleast 0.63 times the optimal solution. Also, it is possible that the solution obtained through CostGreedy

ReduceGreedy (Plan Diagram P, Threshold λ)

1. Let R_1 = CostGreedy (P, λ).
2. Let $k = |\mathsf{R}_1|$.
3. Let R_2 = ThresholdGreedy (P, k).
4. **if** (CostIncrease$(\mathsf{R}_2) >$ CostIncrease(R_1)
   return R_1
   else
   return R_2
5. End Algorithm ReduceGreedy.

Figure A.1: The ReduceGreedy Algorithm

performs better than the one obtained through ThresholdGreedy. Hence, a sanity check is performed in in step 4 to return the best solution.

A.1.2 Results

Tables A.1, A.2 and A.3 show the quality of plan diagram reduction of ReduceGreedy when compared to CostGreedy for three commercial optimizers, denoted by OptA, OptB and OptC respectively. It can be seen that in many situations, using ReduceGreedy helps us obtain a reduced plan diagram with better quality.

Query Template	**No. of Plans**	**Reduced Plans** ($\lambda = 20\%$)	**Maximum λ Reached**	
			CostGreedy	ReduceGreedy
QT2	14	7	5.4%	5.4%
QT5	11	2	17.2%	17.2%
QT8	36	3	18%	16%
QT9	39	6	19.4%	19.4%
QT10	18	4	11%	11%

Table A.1: Performance of ReduceGreedy (OptA)

Query Template	**No. of Plans**	**Reduced Plans** ($\lambda = 20\%$)	**Maximum λ Reached**	
			CostGreedy	ReduceGreedy
QT2	20	8	19.8%	19.8%
QT5	12	4	4.7%	4.7%
QT8	16	2	18.1%	18.1%
QT9	18	3	19.5%	19.5%
QT10	7	3	6%	6%

Table A.2: Performance of ReduceGreedy (OptB)

Query Template	**No. of Plans**	**Reduced Plans** ($\lambda = 20\%$)	**Maximum λ Reached**	
			CostGreedy	ReduceGreedy
QT2	44	8	18.4%	18.4%
QT5	23	5	16.4%	11.6%
QT8	50	4	16%	16%
QT9	38	3	19.4%	12.6%
QT10	17	3	13.8%	13.8%

Table A.3: Performance of ReduceGreedy (OptC)

A.2 Batch Reduction

The plan diagram reduction problem defined in [11] requires the cost increase of all query points in the reduced plan diagram to be less than λ. In this section we will consider a variant of this problem where we require the average cost-increase of all the query points in the reduced plan diagram to be below the given λ-threshold. This problem is formally defined as follows:

Batch Reduction Problem. Given an input plan diagram P, and a maximum-cost-increase threshold λ ($\lambda \geq 0$), find a reduced plan diagram R with *minimum cardinality* such that,

$$\frac{\sum_{q \in \mathsf{R}} c_{re}(q)}{\sum_{q \in \mathsf{R}} c_{oe}(q)} \leq (1 + \lambda)$$

where c_{re} and c_{oe} are the costs of the plans P_{re} and P_{oe} assigned to the query point q in R and P respectively.

A.2.1 The BatchReduce algorithm

It can be easily seen that the batch reduction problem is NP-Hard. Hence, we provide the BatchReduce algorithm, that uses a greedy heuristic to solve this problem.

In this algorithm, we start with the input plan diagram and in each step, we remove a plan P_i such that

1. The total cost increase is within λ-threshold after the removal of P_i, and
2. Removal of P_i has the minimum total cost increase over all plans that can be removed.

The algorithm terminates when no plan can be removed from the plan diagram. The complete algorithm is given in Figure A.2

BatchReduce (Plan Diagram P, Threshold λ)

1. set R = P
2. set $\lambda' = \infty$
3. For each plan $P_i \in$ R
   (a) Calculate λ_i = cost increase of R due to the removal of P_i
   (b) **if** $\lambda_i < \lambda'$
   $\lambda' = \lambda_i, plan = P_i$
4. **if** $\lambda' \leq \lambda$
   (a) remove $plan$ from R
   (b) goto Step 2
5. **return** R
6. End Algorithm BatchReduce.

Figure A.2: The BatchReduce Algorithm

This algorithm does not impose any restrictions on the cost increase of an individual query point. But in some situations, we would like to maintain an upper bound λ_q on the increase in cost of every point. This can be easily achieved by a slight modification to the above algorithm, where in addition to checking the overall cost increase, we also check for the maximum cost increase of any point, when a plan is removed. Another enhancement that can be made is

to use the reduced plan diagram obtained through CostGreedy, instead of P in Step 1 of the algorithm. The advantage of using CostGreedy reduction first is two-fold, (a) it ensures that the reduced plan diagram is the same as the one obtained from CostGreedy when $\lambda_q = \lambda$, and (b) the plans in the reduced plan diagram are likely to be robust since the more volatile plans are usually removed by CostGreedy.

A.2.2 Results

In our experiments, we use the modified BatchReduce algorithm, which uses the reduced plan diagram obtained through CostGreedy in step 1 of the algorithm given in Figure A.2. Tables A.4, A.5 and A.6 show the reduction results for OptA, OptB and OptC respectively, obtained using a representative cost-increase threshold of $\lambda = 20\%$. The reduction was performed for $\lambda_q = \infty$, where there is no upper bound on the cost-increase of an individual query point, and for $\lambda_q = 200\%$, which is an acceptable increase in cost for any given query point.

Query Template	No. of Plans	Reduced Plans	
		$(\lambda_q = \infty)$	$(\lambda_q = 200\%)$
QT2	14	3	5
QT5	11	1	2
QT8	36	1	1
QT9	39	1	1
QT10	18	1	2

Table A.4: Performance of BatchReduce (OptA)

Query Template	No. of Plans	Reduced Plans	
		$(\lambda_q = \infty)$	$(\lambda_q = 200\%)$
QT2	20	1	2
QT5	12	1	2
QT8	16	1	1
QT9	18	1	1
QT10	7	1	2

Table A.5: Performance of BatchReduce (OptB)

Query Template	No. of Plans	Reduced Plans	
		$(\lambda_q = \infty)$	$(\lambda_q = 200\%)$
QT2	44	2	3
QT5	23	2	2
QT8	50	1	1
QT9	38	1	1
QT10	17	1	2

Table A.6: Performance of BatchReduce (OptC)

Bibliography

[1] A. Aboulnaga and S. Chaudhuri, "Self-tuning Histograms: Building Histograms without Looking at Data", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1999.

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

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

[4] S. Babu, P. Bizarro and D. DeWitt, "Proactive Re-Optimization", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, 2005.

[5] S. Babu, P. Bizarro and D. DeWitt, "Proactive Re-Optimization with Rio", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2005.

[6] N. Bruno, "A Critical Look at the TAB Benchmark for Physical Design Tools", *SIGMOD Record*, 36(4), 2007.

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

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

[9] A. Deshpande, Z. Ives and V. Raman "Adaptive Query Processing", *Foundations and Trends in Databases*, 2007.

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

[11] Harish D., P. Darera and J. Haritsa, "On the Production of Anorexic Plan Diagrams", *Proc. of 33rd Intl. Conf. on Very Large Data Bases (VLDB)*, September 2007.

[12] 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.

[13] 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.

[14] Y. Ioannidis and S. Christodoulakis, "On the Propagation of Errors in the Size of Join Results", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 1991.

[15] 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.

[16] E. Kreyszig, *Advanced Engineering Mathematics*, New Age International, 5th ed, 1997.

[17] 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.

[18] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh and M. Cilimdzic, "Robust Query Processing through Progressive Optimization", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, June 2004.

[19] J. Patel, M. Carey and M. Vernon, "Accurate Modeling of the Hybrid Hash Join Algorithm", *Proc. of ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, 1994.

[20] Picasso Database Query Optimizer Visualizer, `http://dsl.serc.iisc.ernet.in/projects/PICASSO/picasso.html`

[21] 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.

[22] 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.

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

[24] 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.

[25] MATLAB, `http://www.mathworks.com`

[26] `http://www.tpc.org/tpch`

[27] `http://www.tpc.org/tpcds`

[28] `http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/t0024533.htm`

[29] `http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.dc34982\_1500/html/mig\_gde/BABIFCAF.htm`

[30] `http://msdn2.microsoft.com/en-us/library/ms189298.aspx`