Reduction of Query Optimizer Plan Diagrams

Harish D. Pooja N. Darera Jayant R. Haritsa

Technical Report
TR-2007-01

Database Systems Lab
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560012, India
http://dsl.serc.iisc.ernet.in

Abstract

A “plan diagram” is a pictorial enumeration of the execution plan choices of a database query optimizer over the relational selectivity space. We have shown recently that, for industrial-strength database engines, these diagrams are often remarkably complex and dense, with a large number of plans covering the space. However, they can often be reduced to much simpler pictures, featuring significantly fewer plans, without materially affecting the query processing quality. Plan reduction has useful implications for the design and usage of query optimizers, including quantifying redundancy in the plan search space, enhancing useability of parametric query optimization, identifying error-resistant and least-expected-cost plans, and minimizing the overheads of multi-plan approaches.

We investigate here the plan reduction issue from theoretical, statistical and empirical perspectives. Our analysis shows that optimal plan reduction, w.r.t. minimizing the number of plans, is an NP-hard problem in general, and remains so even for a storage-constrained variant. We then present a greedy reduction algorithm with tight and optimal performance guarantees, whose complexity scales linearly with the number of plans in the diagram for a given resolution. Next, we devise fast estimators for locating the best tradeoff between the reduction in plan cardinality and the impact on query processing quality. Finally, extensive experimentation with a suite of multi-dimensional TPC-H and TPC-DS based query templates on industrial-strength optimizers demonstrates that complex plan diagrams easily reduce to “anorexic” (small absolute number of plans) levels incurring only marginal increases in the estimated query processing costs.

1 Introduction

Modern database systems use a *query optimizer* to identify the most efficient strategy to execute declarative SQL queries. The efficiency of the strategies, called "plans", is usually costed in terms of the estimated query response time. Optimization is a mandatory exercise since the difference between the cost of the best plan and a random choice could be in orders of magnitude [25]. The role of query optimizers has become especially critical in recent times due to the high degree of query complexity characterizing current decision-support applications, as exemplified by the TPC-H benchmark [26], and its recent incarnation, TPC-DS [27].

Plan Diagrams

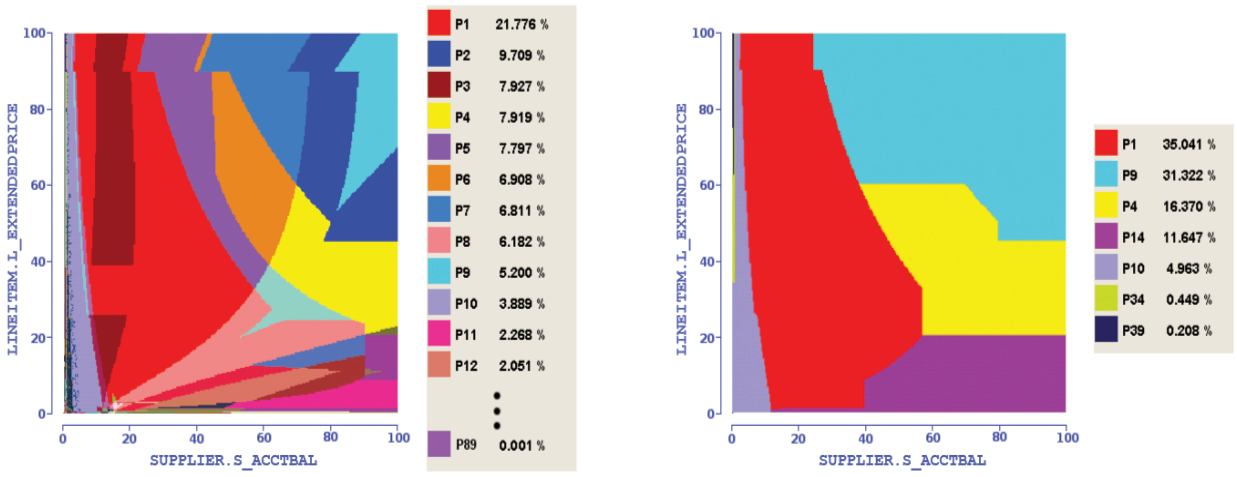
For a query on a given database and system configuration, the optimizer's plan choice is primarily a function of the *selectivities* of the base relations participating in the query – that is, the estimated number of rows of each relation relevant to producing the final result. In a recent paper [19], we introduced the concept of a "plan diagram" to denote a color-coded pictorial enumeration of the execution plan choices of a database query optimizer for a parameterized query template over the relational selectivity space. For example, consider QT8, the parameterized 2-D query template shown in Figure 1, based on Query 8 of the TPC-H benchmark, with selectivity variations on the SUPPLIER and LINEITEM relations through the s_acctbal :varies and l_ extendedprice :varies predicates, respectively. The associated plan diagram for QT8 is shown in Figure 2(a), produced with the Picasso query optimizer visualizer tool [17] on a popular commercial database engine.[1]

```
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: **Example Query Template: QT8**

In Figure 2(a), the X and Y axes determine the percentage selectivities of the SUPPLIER and LINEITEM relations, respectively, and each color-coded region represents that a particular plan has

[1] Plan diagrams can be computationally expensive to produce but such investments are likely to be acceptable for canned query templates, like those found in Web applications.



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

Figure 2: **Sample Plan and Reduced Plan Diagrams (QT8)**

been determined by the optimizer to be the optimal choice in that entire region. We find that a set of 89 different optimal plans, P1 through P89, cover the entire selectivity space. The value associated with each plan in the legend indicates the percentage ara coverage of that plan in the diagram – P1, for example, covers about 22% of the space, whereas P89 is chosen in only 0.001% of the space.
[*Note to Readers: We request the readers to view the plan diagrams directly from the color PDF file, rather than from a print copy since the grayscale version may not clearly register the various features.***]**

Anorexic Plan Diagrams

As is evident from Figure 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 query templates based on the TPC-H benchmark, over a suite of commercial optimizers, are available at [17]. However, we had also shown in [19] that these dense diagrams could typically be "reduced" to much simpler pictures featuring significantly fewer plans, *without adversely affecting the query processing quality*.

For example, if we were willing to tolerate a minor cost increase of at most 10% for any query point in the diagram relative to its original (optimizer-estimated) cost, Figure 2(a) could be reduced to that shown in Figure 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. Further, note that a 10% increase, apart from being small in absolute terms, is also well within the bounds of the *inherent* error that characterizes the estimation process of modern optimizers [14, 20, 24]. The graph of the reduced diagram's plan cardinality as a function of the cost increase threshold for this example is shown in Figure 3.

In general, our experience over a wide spectrum of dense plan diagrams ranging from tens to hundreds of plans, across the optimizer suite, has been that a cost increase threshold of *only twenty percent*

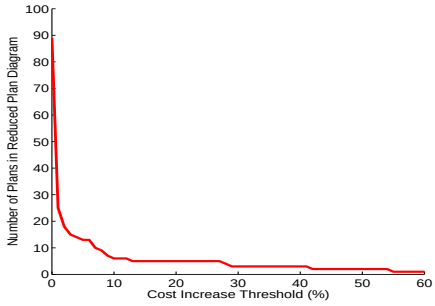


Figure 3: Plan Cardinality vs Cost Threshold

is amply sufficient to bring down the number of plans in the final reduced picture to *within ten*. In short, that plan diagrams can usually be made "anorexic" while retaining acceptable query processing performance.

Further, as we will show in detail later in this report, it is possible to achieve this reduction efficiently since we limit our attention to only the set of plans appearing in the original plan diagram, and do not revisit the exponentially large search space of plan alternatives from which the optimizer makes these choices.

We hasten to add that while our focus is primarily on selectivity-based parameterization in this report, the above observations on the reduction behavior also hold true for parameterization on the *memory* made available to the database engine for query processing, a key factor impacting optimizer plan choices [5].

Contributions

We consider here the problem of reducing plan diagrams, from theoretical, statistical and empirical perspectives. We first show that finding the optimal (w.r.t. minimizing the plan cardinality) reduced plan diagram is NP-Hard through a reduction from Set Cover. This result motivates the design of CostGreedy, a greedy algorithm whose complexity is $O(nm)$, where n is the number of plans and m is the number of query points in the diagram $(n \ll m)$. Hence, for a given picture resolution, CostGreedy's performance scales *linearly* with the number of plans in the diagram, making it much more efficient than the $O(m^2)$ reduction algorithm of [19]. Further, from the reduction quality perspective, CostGreedy provides a tight performance guarantee of $O(\ln m)$, which cannot be improved upon by any alternative deterministic algorithm.

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

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

Last, through an experimental analysis on the plan diagrams produced by industrial strength optimizers with TPC-H and TPC-DS based multi-dimensional query templates, we show that (a) plan

reduction can be carried out efficiently, (b) the CostGreedy algorithm typically gives the optimal reduction or is within a few plans of the optimal, (c) the analytical estimates of the plan-reduction versus cost-threshold curve are quite accurate, and finally, that (d) a 20% cost threshold is amply sufficient to bring the plan cardinality to within or around 10, even for high dimensional query templates – this is an especially promising result from a practical perspective.

2 Anorexic Reduction Benefits

The production of anorexic reduced plan diagrams, that is, diagrams whose plan cardinality is within/around a small absolute number (10 is the yardstick used here), has a variety of useful implications for improving both the efficiency of the optimizer and the choice of execution plan, as outlined below:

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

Enhancement of PQO Usability: A rich body of literature exists on *parametric query optimization* (PQO) (e.g.[5, 12, 13, 8, 9, 15]). The goal here is to apriori identify the optimal set of plans for the entire relational selectivity space at compile time, and subsequently to use at run time the actual selectivity parameter settings to identify the best plan – the expectation is that this would be much faster than optimizing the query from scratch.

A practical difficulty with PQO, however, is the representation of the plan optimality boundaries, which could, in principle, be of arbitrary complexity, making it difficult to identify specifically which plan from the set of optimal plans is to be utilized for a newly arrived query. A workaround for this problem is the following [13]: For the specific query currently supplied by the user, evaluate its estimated execution cost with *each of the plans* in the optimal set. Then, choose the lowest cost plan for executing the query. For this workaround to be viable, the plan diagram must have, in an absolute sense, only a small number of plans – this is because while plan-costing is cheap as compared to query optimization [13], the total time taken for many such costings may become comparable. However, as shown in Figure 2(a), the number of optimal plans can be very large, unless plan reduction is applied.

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

Identification of Error-Resistant Plans: Plan reduction can help to identify plans that provide robust performance over large regions of the selectivity space. Therefore, *errors* in the underlying database statistics, a situation often encountered by optimizers in practice [14], may have much less impact as compared to using the fine-grained plan choices of the original plan diagram, which may have poor performance at other points in the space.

For example, in Figure 2(a), estimated selectivities of (14%,1%) leads to a choice of plan P70. However, if the actual selectivities at runtime turn out to be significantly different, say

(50%,40%), using plan P70, whose cost increases steeply with selectivity, would be disastrous. In contrast, this error would have had no impact with the reduced plan diagram of Figure 2(b), since P1, the replacement plan choice at (14%,1%), remains as the preferred plan for a large range of higher values, including (50%,40%). Quantitatively, at (50%, 40%), plan P1 has a cost of 135, while P70 is much more expensive, about *three times* this value.

In short, the final plan choices become robust to errors that lie within the optimality regions of the replacement plans. Such stability of plan choices is especially important for industrial workloads where often the goal is to identify plans with stable good overall performance as opposed to selecting the best local plan with potentially risky performance characteristics [16].

Identification of Least-Expected-Cost Plans: When faced with unknown input parameter values, today's optimizers typically approximate the distribution of the parameter values using some representative value – for example, the mean or modal value – and then always choose this "least specific cost" plan at runtime. It has been shown in [3, 4] that a better strategy would be to instead optimize for the "least expected cost" plan, where the full distribution of the input parameters is taken into account. Computing the least expected cost plan not only involves substantial computational overhead when the number of plans is large, but also assumes that the various plans being compared are all modeled at the same level of accuracy, rarely true in practice. With plan reduction, on the other hand, both the efficiency and the quality of the comparisons can become substantially better since there are fewer contending plans.

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

An alternative and less resource-intensive multi-plan approach is proposed in [14] wherein during execution of the best compile-time plan choice, based on the observed run-time performance, a change in the query plan could be triggered for the remaining unexecuted portion of the query. When this approach is combined with plan reduction, the likelihood of triggering a re-optimization becomes substantially lower, thereby reducing the associated overheads.

Supports Plan Clustering: Plan reduction fits in perfectly with the query clustering approach previously proposed in our Plastic plan recycling tool [7, 21, 22, 29], where queries that are expected to have identical plan templates are grouped together based on similarities in their feature vectors. This is because the cluster regions *inherently* coarsen the plan diagram granularity. Further, from an implementation perspective, having fewer distinct plans makes it easier with regard to both storage and comparison.

Picasso Execution Diagram Time Estimation: Apart from producing compilation diagrams, the Picasso tool [17] also supports the production of execution cost diagrams which show the actual run-time costs of executing the query points in the plan diagram. As a precursor to this process, the user is given an estimate of the time taken to produce the entire picture, and this is achieved

by first fully executing a sample query point and then extrapolating its response time to the sum of the optimizer-estimated costs of the remaining query points. For the sample query point, we would like to ideally choose, from an efficiency and representativeness perspective, the cheapest query point associated with the plan that occurs most frequently in the plan diagram. In the reduced plan diagram, it is likely that we will find a much cheaper point represented by this most frequent plan since the areas covered by the surviving plans increase substantially.

3 Related Work

To the best of our knowledge, apart from the initial results presented by us in [19], there has been no prior work on the reduction of plan diagrams with regard to *real-world industrial-strength* query optimizers and query templates. However, similar issues have been considered in the PQO literature in the context of simplified optimizers and basic query workloads. Specifically, in the pioneering work of Betawadkar & Ganguly [2], a System-R style optimizer with left-deep join-tree search space and linear cost models was built, the workload comprising of pure SPJ query templates with star or linear join-graphs and one-dimensional selectivity variations. Within this context, their experimental results indicate that, for a given cost increase threshold, plan reduction is more effective with increasing join-graph complexity. They also find that "if the increase threshold is small, a significant percentage of the plans have to be retained." For example, with a threshold of 10%, more than 50% of the plans usually have to be retained. However, this conclusion is possibly related to the low plan cardinality (less than 20 in all the experiments) in their original plan diagrams. In contrast, our results indicate that on the dense plan diagrams seen in real-world environments, where the number of plans can be in the hundreds, not only is the reduction very substantial even for a 10% cost increase, but even more strikingly, that the reduced plan cardinality is small in *absolute terms*.

In the subsequent work of [12, 13], Hulgeri & Sudarshan model an optimizer along the lines of the Volcano query engine [11], and evaluate SPJ query templates with two, three and four-dimensional relational selectivities. In their formulation, the cost increase threshold cannot be guaranteed in the presence of nonlinear cost functions, a common feature in practice, and is used only as a heuristic. Even with this relaxation, the final number of plans with a threshold of 10% can be large – for example, a 4-D query template with 134 original plans is reduced only to 53 with the DAG-AniPOSP algorithm and to 29 with AniPOSP. Our work differs in that (a) we guarantee to maintain the cost increase threshold, and (b) the observed reductions are substantially higher.

Finally, we provide for the first time, efficiency and quality guarantees for the reduction algorithms, as well as cardinality estimators for the reduced plan diagram.

4 The Plan Reduction Problem

In this section we define the Plan Reduction Problem, hereafter referred to as PlanRed, and prove that it is NP-Hard through a reduction from the classical Set Cover Problem [10]. For ease of exposition, we assume in the following discussion that the source SQL query template is 2-dimensional – the extension to higher dimensions is straightforward.

4.1 Preliminaries

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

Definition 1 *Plan Diagram*

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

1. *Each point* $q(x, y)$ *in the grid corresponds to a unique query with (percentage) selectivities* x, y *in the X and Y dimensions, respectively.*
2. *Each query point* q *in the grid is associated with an optimal plan* P_i *(as determined by the optimizer), and a cost* $c_i(q)$ *representing the estimated effort to execute* q *with plan* P_i*.*
3. *Corresponding to each plan* P_i *is a unique color* L_i*, which is used to color all the query points that are assigned to* P_i*.*

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

With the above framework, PlanRed is defined as follows:

Definition 2 *PlanRed*

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

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

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

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

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

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

Intuitively, what the PCM condition states is that we expect the query execution cost of a plan to increase with base relation selectivities. For most query templates, this is usually the case since an increase in selectivity corresponds to processing a larger amount of input data. However, the assumption may not hold for query templates that feature negation operators such as "set difference", or short-circuit operators like "exists" – we discuss how to handle such situations in A.1. For the remainder of this report, we consider only the common case of plan diagrams in which the PCM condition applies.

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

Definition 3 *Cost Bounding Rule*

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

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

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

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

4.2 The Set Cover Problem

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

Definition 4 *Set Cover Problem*

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

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

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

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

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

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

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

4.3 Reducing Set Cover to PlanRed

We now show that the Set Cover problem can be reduced to the Plan Reduction problem. Specifically, Algorithm Reduce in Figure 4 converts an instance of Set Cover to an instance of PlanRed. It takes as input the instance I' and threshold λ and outputs a plan diagram and another instance $I_{new} = (U_{new}, S'_{new})$ of Set Cover.

The data structures used in the algorithm are as follows:

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

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

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

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

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

Algorithm Reduce works as follows: Consider a Set Cover instance $I' = (U', S_{new})$. For each subset $S_i \in S_{new}$, a unique color L_i which represents the plan P_i is created. Each element $q \in U'$ represents a query point in P, and let q be in subsets $S_{i_1}, S_{i_2}, \ldots S_{i_k}$ for each $S_{i_j} \in S_{new}$, $j = 1, 2, \ldots k$ and $i_1 < i_2 < \ldots < i_k$. Plan P_{i_1} is chosen as the representative for q and becomes the plan with which q is associated. For each of the other subsets in which q is present, a new query point r is created and placed to the right of q in the plan diagram, with its color corresponding to the subset it represents and its cost being $(1 + \lambda)$ times the cost of q. Intuitively this means that plan P_{i_1} can be replaced by plans $P_{i_j}, j = 2, 3 \ldots k$. Then, a query point t is created having plan P' corresponding to the subset S' with a cost $(1 + \lambda)^2$ times the cost of q – this point is added to the right of all the points that were previously created for q. This means that t can in turn replace all the other points that were created for q, but not q itself. (Note that this process is identical to the element addition process of Lemma 1.) When moving from the last element of one row to the first element of the next row, the cost is further increased by a factor of $(1 + \lambda)$.

Reduce(Set Cover I')

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

Figure 4: Algorithm Reduce

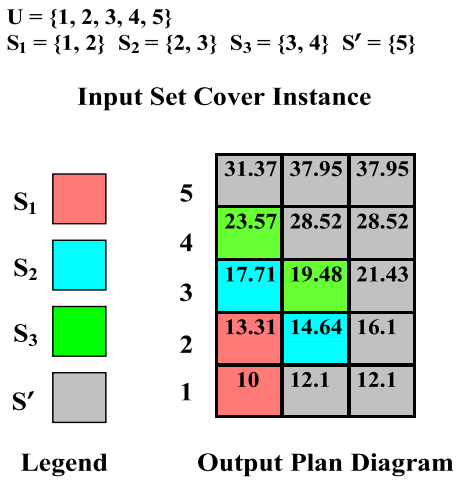


Figure 5: Example of Algorithm Reduce

Starting from the bottom row and moving upwards, the above procedure is repeated for each element, resulting in each element and its associated generated points being assigned to different rows in the plan diagram. Finally, for each empty point in the grid, a new query point q is created having plan P' corresponding to the subset S' with a cost equal to the cost of the rightmost point in its row with the plan P'. An example of this reduction, with $\lambda = 10\%$, is shown in Figure 5, where each point is represented by a square block. The blocks in the first column of the output plan diagram represent the elements originally in U, while the remaining blocks are added during the reduction process. The values in the blocks represent the costs associated with the corresponding points, and each subset is associated with a color, as shown in the legend.

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

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

Proof:

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

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

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

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

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

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

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

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

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

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

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

Proof: It can be seen that

1. Algorithm Reduce has a polynomial time complexity of $O(nm)$.
2. For $I' = (U', S_{new})$, the grid created has in the worst case $|U'| * (|S_{new}|)$ elements with $|S_{new}|$ plans. It is a valid plan diagram. (Lemma 2)
3. The optimal solution for Set Cover Instance I' can be obtained by the optimal solution of the plan diagram generated by the algorithm. (Lemma 3)

Hence the theorem. ■

In the hope of finding a polynomial-time optimal solution we also considered a situation where, rather than allowing a plan to be collectively swallowed by a group of sibling plans, we mandate that a plan can be swallowed only if it can be entirely replaced by a *single* sibling plan. That is, all query points of a swallowed plan have the identical replacement color. Unfortunately, however, this constraint does not change the complexity of the problem, as proved in Appendix A.2.

4.4 Storage-budgeted Plan Reduction

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

Definition 5 *Storage-budgeted Plan Reduction Problem*
Given a plan diagram P *and storage constraint of retaining at most k plans, find the k plans to be chosen so as to minimize the maximum cost increase of the query points in the reduced plan diagram* R.

A Karp Reduction [10] can be used to show that Storage-budgeted PlanRed is NP-Hard by using it to solve the general Plan Reduction problem, leading to the following theorem (Proof in A.3):

Theorem 2 *The Storage-budgeted Plan Reduction Problem is NP-Hard.*

5 Greedy Plan Reduction

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

We first consider AreaGreedy, the reduction algorithm proposed in [19], where the greedy heuristic is based on plan areas. Then we present CostGreedy, a new reduction algorithm that is greedy on plan costs. Its computational efficiency and reduction quality guarantees are quantified for the general PlanRed. We then present a greedy algorithm ThresholdGreedy that has strong performance bounds for the storage-budgeted version. As before, for ease of exposition, we assume that the input plan diagram is 2-dimensional – the algorithms can be easily generalized to higher dimensions, while the theoretical results are independent of the dimensionality.

5.1 The AreaGreedy Algorithm

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

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

The intuition behind the design of AreaGreedy is two-fold: First, using an area basis for the swallowing iterations is likely to reduce the number of small-sized plans. This would contribute towards plan stability as discussed in the Introduction. Second, small-sized plans tend to be found near the origin and the axes of the plan diagram [18, 13, 19] – this means that they offer more scope for swallowing since their first quadrants are big and therefore likely to have many more candidate swallower plans as compared to the larger-sized plans which occur in the higher regions of the selectivity space. The algorithm is given in A.4

By inspection, it is obvious that AreaGreedy has a time complexity of $O(m^2)$, where m is the number of query points in the plan diagram. With respect to reduction quality, let AG denote the solution obtained by AreaGreedy, and let Opt denote the optimal solution. We have shown in A.5 that the upper bound of the approximation factor $\frac{|AG|}{|Opt|}$ will be atleast $0.5\sqrt{m}$.

5.2 The CostGreedy Algorithm

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

Consider an instance of PlanRed that has an $m_1 \times m_2$ grid with n plans and $m = m_1 \times m_2$ query points. By scanning through the grid, we can populate the cur and $belong$ data structures (introduced in Section 4.3) for every point. This can be done as follows: For each query point q with plan P_i in

the grid, set $cur(q)$ to be i, and add to $belong(q)$ all j such that P_j can replace q. Using this, a Set Cover instance $I = (U, S)$ can be created with $|U| = m$ and $|S| = n$. Here U will consist of elements that correspond to all the query points and S will consists of sets corresponding to the plans in the plan diagram. The elements of each set will be the set of query points that can be associated with the plan corresponding to that set.

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

Lemma 4 *The optimal solution of the created Set Cover instance I gives the optimal reduction solution to the plan diagram* P *that is used to create the instance.*

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

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

Lemma 4 is explicitly used in the design of CostGreedy, shown in Figure 6. In Lines 1 through 6, an instance $I = \{U, S\}$ of Set Cover is created. Then, in Line 8, CostGreedy calls Algorithm Greedy Setcover, shown in Figure 7, which takes this input instance and outputs the cover $C \subseteq S$.

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

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

5.2.1 Complexity Analysis

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

Theorem 3 *The time complexity of CostGreedy is $O(mn)$, where m and n are the number of query points and plans, respectively, in the input plan diagram* P.

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

CostGreedy (Plan Diagram P, Threshold λ)

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

Figure 6: CostGreedy

Greedy Setcover(Set Cover I)

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

Figure 7: Algorithm Greedy Setcover

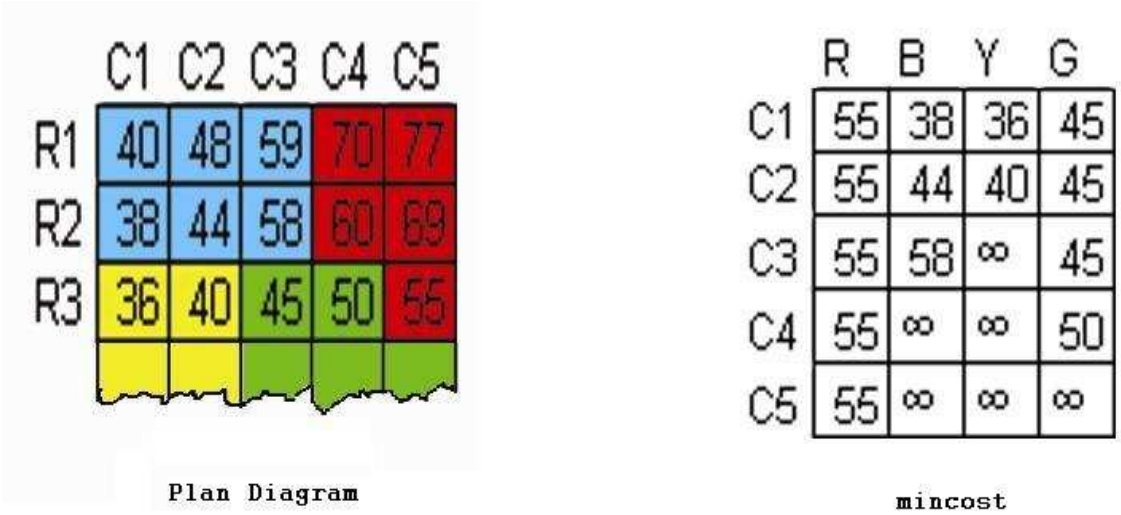


Figure 8: **Updating *mincost* in Algorithm CostGreedy**

We start the scan of the grid from right to left, beginning with the top row of the grid. For each point q with plan P_k at column i in the current row, if it can be replaced by any other plan P_j, then $mincost[i][P_j]$ should be within the increase threshold of the cost of q. Hence, through a single scan of $mincost[i]$, we can populate $belong(q)$. Then the cost of q is updated for $mincost[i][P_k]$. Since the values in the column $mincost[i]$ are candidates for the minimum values of the column $i-1$, $mincost[i-1]$ is updated with the value $min(mincost[i], mincost[i-1])$. An example is shown in Figure 8. The array $mincost$ contains updated values after processing all the columns of the first three rows of the plan diagram.

With the above procedure, when moving to the next row to be processed, the columns $mincost[i]$ will automatically contain the minimum costs of all the plans appearing in the first quadrant of the query point at the i^{th} column of the previous row. When a query point at column i is being processed, due to the cumulative updation of the costs of the plans visited on that row, $mincost[i]$ will be updated with the minimum costs of all the plans in that point's first quadrant.

So each query point requires $2n$ iterations to be made, and there are m query points. Hence the time required for populating the data structures cur and $belong$ is of the order $O(mn)$.

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

5.2.2 Approximation Factor

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

Lemma 5 *CostGreedy has an approximation factor* $\frac{|CG|}{|Opt|} = O(\ln m)$*, where* m *is the number of query points in the plan diagram.*

Proof: It has been shown in [6, 23] that Algorithm Greedy Setcover (GS) has an approximation factor $\frac{|GS|}{|Opt|} \leq H(m)$, where m is the cardinality of the universal set, and $H(m)$ is the m^{th} harmonic number. The input to GS can have at most $(m-1)$ elements in its universal set (this occurs when the TopRight query point has a unique color not shared by any other point in the entire diagram). Therefore,

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

■

Tightness of Bound. It is shown in [23] that given any k, l where $|Greedy| = k$ and $|Opt| = l$, a Set Cover instance can be generated with $(k+l)$ sets and m elements such that $m \geq G(k,l)$, where $G(k,l)$ is a recursively defined greedy number:

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

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

It is also shown in [23] that the following tight bound of $\ln m$ for Set Cover can be achieved using such a construction when $m = G(k,l)$:

$$\ln m - \ln\ln m - 0.31 \leq \frac{k}{l} \leq \ln m - \ln\ln m + 0.78 \tag{2}$$

These results are used in the following lemma.

Lemma 6 *The bound specified by Lemma 5 is tight.*

Proof: The construction process in [23] of the above-mentioned Set Cover instance, with $m = G(k,l)$, is such that every element belongs to exactly *two* sets. For a given k, l, first construct the Set Cover instance using the construction in [23]. Using this create another Set Cover instance of the form I' with $(k+l+1)$ sets and $(m+1)$ elements, as mentioned in Section 4.2. When Algorithm Reduce is applied to this new instance, it creates a grid with $m' = 3*(m+1)$ elements. This is because, for each element, since it is in two sets, it can be colored by two colors in the plan diagram. One of these will represent its current plan, and for the other plan, a new element will be created and added to its right. Then another element will be created to its right which can replace this newly created element and having the color representing the plan corresponding to the set S'. Hence, each of the $m+1$ rows will have 3 elements.

From Equation 2 we know that

$$\frac{|Greedy|}{|Opt|} \geq \ln m - \ln\ln m - 0.31 \tag{3}$$

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

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

■

Optimality of the Bound. It has been shown in [6] that the bound of $O(\ln m)$ for Set Cover is the best possible bound below which Set Cover cannot be approximated efficiently, unless NP has slightly super-polynomial-time algorithms. This result is used in the following theorem:

Theorem 4 *The bound specified by Lemma 5 is the best possible threshold below which PlanRed cannot be approximated efficiently unless NP has slightly super-polynomial-time algorithms.*

Proof: Assume that there exists some deterministic algorithm, DetX, that improves on the bounds of $O(\ln m)$ for PlanRed. Then, for the instance of the grid created from a Set Cover instance, we will have a reduced bound. This means we can get a reduced bound on Set Cover by reducing it into a plan diagram and applying DetX to it. But this would contradict the result of [6]. ■

5.3 The ThresholdGreedy Algorithm

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

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

We now bootstrap the selection algorithm by choosing this plan and subsequently choose additional plans based on their relative benefits. The details of the algorithm can be found in Figure 16. Let Ben_{opt} and Ben_{greedy} be the total benefit of choosing k plans by the optimal and greedy algorithms, respectively. This means that the final cost increase threshold with the optimal selection is $\lambda_{SinPlan} - Ben_{Opt}$, and with the threshold greedy solution is $\lambda_{SinPlan} - Ben_{TG}$. The following theorem quantifies the approximation factor of ThresholdGreedy (proof in A.6)

Theorem 5 *Given a storage budget of k plans, let Ben_{opt} be the benefit obtained by the optimal solution's selection, and Ben_{TG} be the benefit obtained by the ThresholdGreedy algorithm's selection. Then*

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

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

6 Estimators for Plan Reduction

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

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

Our first estimator, AvgEst, takes as input the plan diagram P and a cost increase threshold λ, and returns the estimated number of plans in the reduced plan diagram R obtained with that threshold. It uses the average of the costs of all the query points associated with a plan, to summarize the plan's cost distribution. All these averages can be simultaneously computed with a single scan of the Plan Diagram. AvgEst then sets up an instance of Set Cover, as shown in Figure 9, with the number of elements equal to the number of plans, and the set memberships of plans is based on their representative average costs satisfying the λ threshold. On this instance, the Greedy Set Cover algorithm, introduced earlier in Figure 7, is executed. The cardinality of the solution is returned as an estimate of the number of plans that will feature in R.

AvgEst (Plan Diagram P, Threshold λ)

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

Figure 9: Algorithm AvgEst

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

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

7 Experimental Results

Having considered the theoretical and statistical aspects of plan diagram reduction in the previous sections, we now move on to presenting our experimental results. The testbed is the Picasso optimizer visualization tool [17], executing on a Sun Ultra 20 workstation equipped with an Opteron Dual Core 4GHz processor, 4 GB of main memory and 240 GB of hard disk, running the Windows XP Pro operating system. Through the GUI of the Picasso tool, users can submit a query template, the grid resolution and distribution at which the instances of this template should be spread across the selectivity space, the parameterized relations (axes) and their attributes on which the diagrams should be constructed, and the choice of query optimizer. With this information, the tool automatically generates the associated SQL queries, submits them to the optimizer to generate the plans, and finally produces the color-coded plan, cost and cardinality diagrams.

We conducted our plan reduction experiments over dense plan diagrams produced from a variety of multi-dimensional TPC-H and TPC-DS based query templates evaluated over a suite of industrial-strength database query optimizers. The templates were instantiated at a variety of grid resolutions, based on the experimental objectives and ensuring viable diagram production times. We also confirmed that all the plan diagrams were in compliance with the plan cost monotonicity condition, described in Section 4.1.

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

7.1 Computational Efficiency

We start off by first quantitatively evaluating the runtimes of the two greedy algorithms, AreaGreedy [19] and CostGreedy (proposed in this thesis), as compared to the time taken to produce the computationally-hard optimal solution. The reduction quality of the algorithms is compared in the next section. A sample set of results on OptC is shown in Table 1 for QT8, the query template shown in Section 1, instantiated at a grid resolution of 100 uniformly distributed points per dimension[2] and reduction carried out at a cost increase threshold of 10%. We see here that even for this relatively coarse-grained situation, the optimal algorithm takes several hours to complete. In contrast, AreaGreedy takes only a few seconds, while CostGreedy is an order-of-magnitude better than AreaGreedy, finishing in a small fraction of a second.

The substantial improvement of CostGreedy with regard to AreaGreedy is, as per the discussion in Section 5, due to its $O(nm)$ complexity being significantly lower than the $O(m^2)$ of AreaGreedy, as

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

Table 1: Computational Efficiency (QT8, Res=100)

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

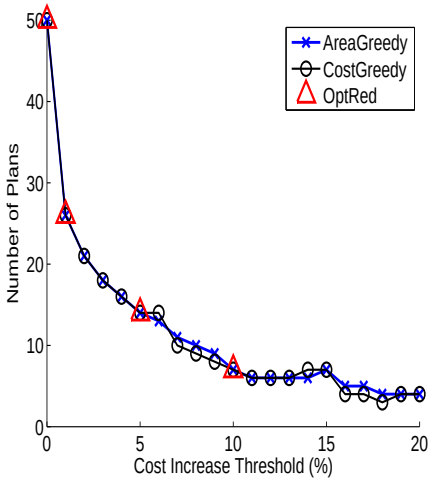


Figure 10: Reduction Quality (QT8), Res=100

$n \ll m$ in practice (recall that n is the number of plans and m is the total number of query points in the plan diagram).

7.2 Plan Reduction Quality

Turning our attention to the reduction quality, we see in Table 1 that AreaGreedy and CostGreedy are identical to the optimal (OptRed), all three producing reduced plan diagrams with 7 plans (in fact, the plans themselves are also the same in this case). The closeness to the optimal holds across the entire operational range of cost increase thresholds, as shown in Figure 10, which presents the reduced plan cardinalities for the three algorithms as a function of the threshold – only a few representative points were obtained for OptRed due to its extremely high computational overheads.

Another point to note in Figure 10 is the initial steep decrease in the number of plans with increasing threshold – we have found this to be a staple feature of all the dense plan diagrams that we have investigated, irrespective of the specific query template, data or query point distribution, memory availability, or database optimizer that produced the dense diagram. These settings may determine *whether or not* a dense plan diagram is produced, but if produced, subsequently the reduction process produces consistent results. This trend is clearly seen in Table 2, which captures the reduction behavior of Optimizers A, B and C, with various TPCH-based query templates on which they produced dense plan diagrams.

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

Table 2: Plan Reduction Quality (Res = 100)

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

Table 3: Skewed Data Distribution (Res = 100)

7.3 Skewed Data Distribution

The above results were obtained with uniformly distributed data generated using the TPC-H benchmark's synthetic generator. We extended the generator to also produce skewed data distributions. When this skewed data was used instead, the observed reduction results did not materially change. While the specific plan diagram changed, the reduction behavior continued to be as before. This can be seen in Table 3, which captures the behavior of the three optimizers on their dense plan diagrams with skewed data.

7.4 Exponential Distribution of Query Points

In the above diagrams, which were produced with a uniform distribution of query points across the selectivity space, we observed that in most cases, the density of plans is greater in the regions near the axes, that is, at low selectivity values of the base relations. This motivated us to alter the arrangement of query points to be exponentially distributed with a higher density in the low selectivity region. As expected, this led to a substantial increase in the cardinality of the original plan diagram. Despite this, we see that the reduction process remains materially unaffected. This is highlighted in Table 4, where we see that the plan cardinality of the reduced plan diagram decreases sharply at a low cost increase threshold, irrespective of the number of plans in the original plan diagram. For example, the plan diagram cardinality increased from 38 to 225 for QT9 on OptC, but the reduced plan diagram

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

Table 4: Exponential Query Point Distribution (Res = 100)

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

Table 5: Increased Grid Resolution (Res = 300)

cardinality (with $\lambda = 20\%$) went from 3 plans to only 8 plans.

7.5 Increased Grid Resolution

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

This is clearly seen in Table 5, which captures the reduction behavior of the three optimizers with the TPCH-based query templates at a grid resolution of 300 uniformly distributed query points per dimension. For example, although the plan diagram cardinality went up from 38 to 91 in case of QT9 on OptC, the reduced plan diagram cardinality (with $\lambda = 20\%$) went from 3 plans to only 4 plans. This means that for practical threshold settings, the final plan cardinality in the reduced diagram is essentially "scale-free" with regard to resolution.

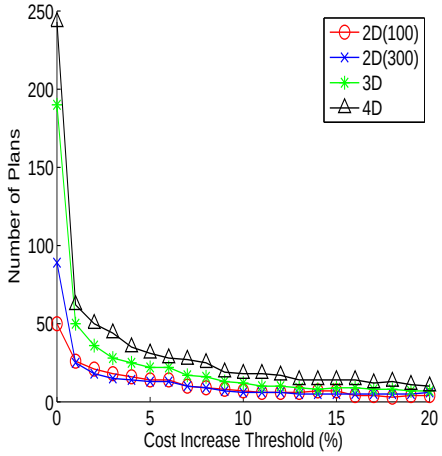


Figure 11: Scaling with Dimensions

Table 6: Multi-dimensional Query Templates

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

7.6 Scaling with Dimensions

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

The results are shown in Figure 11 for 2-D with resolutions of 100 and 300 query points per dimension, 3-D with a resolution of 100 query points per dimension, and 4-D with a resolution of 30 query points per dimension. We see here that while the number of plans in the original plan diagram goes up steeply with increasing dimensionality, the reduction behavior is qualitatively similar across all the templates. Further, as shown in Table 6, the reduction behavior is remarkably stable: First, the location of the knee of the plan cardinality vs. cost increase threshold graph varies only marginally, occurring in the neighborhood of 10%. Second, the threshold required to bring the reduced plan diagram cardinality down to 10 plans is within 20%, a very practical value from a user perspective, even in a 4-D setting. Again, this seems to suggest that for practical threshold settings, the final plan cardinality in the reduced plan diagram is essentially “scale-free” with regard to dimension.

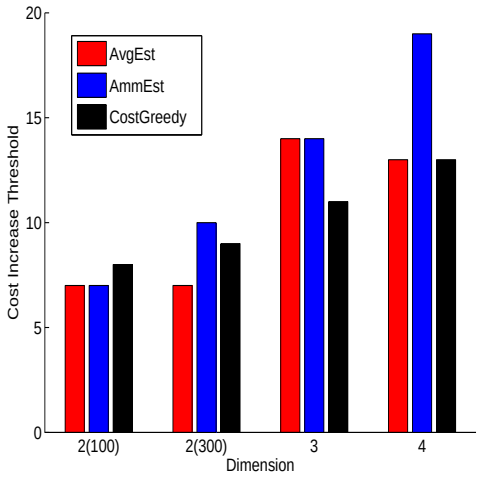


Figure 12: Knee Estimates

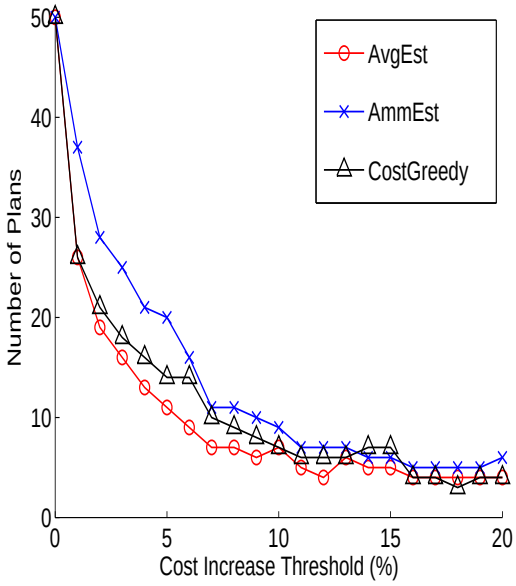
7.7 Estimator Performance

Our next experiment studies the quality of the *knee estimates* provided by the estimators. The results are shown in Figure 12 for QT8 on OptC (the results for other query templates and database engines are similar in nature) and indicate that AvgEst and AmmEst are reasonably accurate despite using extremely coarse characterizations of the cost distributions of plans in their optimality regions. Further, their orders-of-magnitude runtime efficiency relative to the CostGreedy algorithm, for iteratively computing the knee, is captured in Table 7.

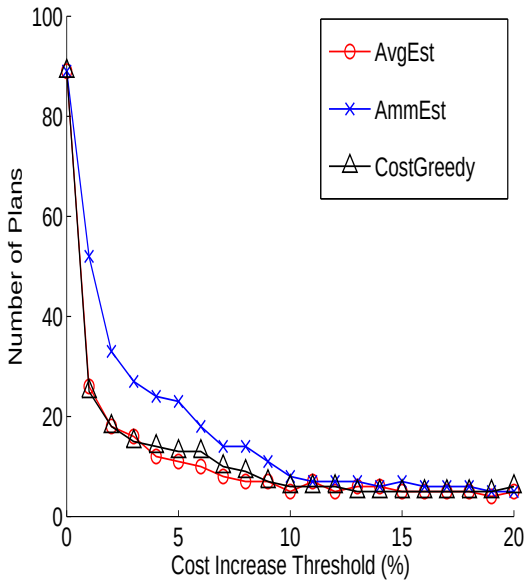
The estimator performance in characterizing the full plot of reduced plan cardinality versus λ is shown in Figures 13(a)–13(d) for 2D-100, 2D-300, 3D-100 and 4D-30, respectively, the CostGreedy performance being used as the yardstick. We see here that, in general, the simple AvgEst estimator provides estimates that are closer to CostGreedy than AmmEst– however, an advantage of AmmEst is that it produces *conservative* estimates, whereas AvgEst can on occasion slightly overestimate the degree of plan reduction, as is seen in Figures 13(a) and 13(b).

Table 7: Running Time of Estimators vs CostGreedy

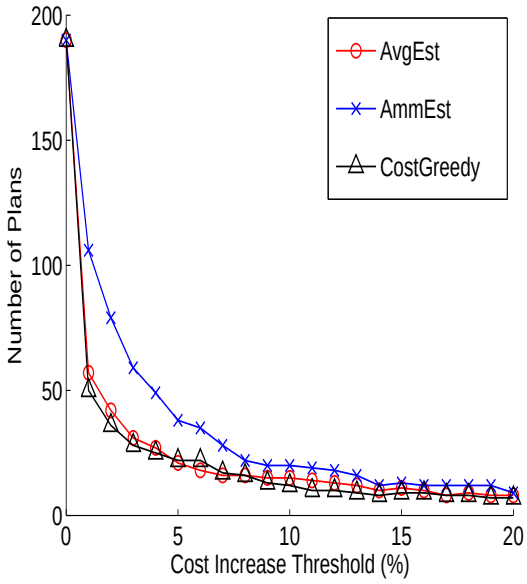
TPC-H Query Template	**Estimator Time(ms) (for Knee)**	**CostGreedy time(ms) (for Knee)**
2	25	2733
5	8	1675
8	26	3648
9	71	2382
10	12	546



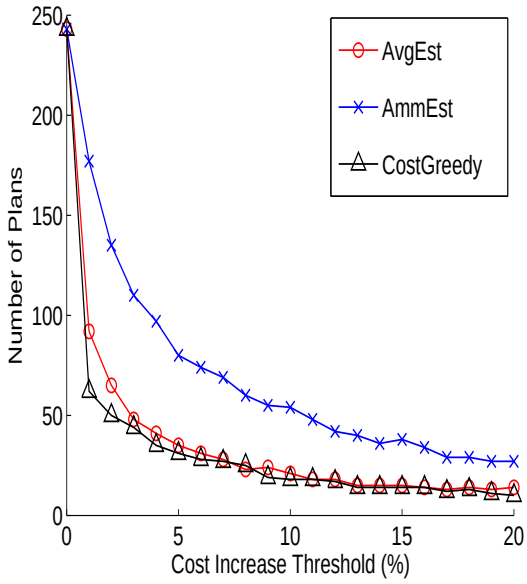
(a) **Est-2D (100)**



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



(c) **Est-3D**



(d) **Est-4D**

Figure 13: Estimator Performance

Table 8: OptA- Varying memory

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

7.8 Effect of Memory Availability

In all the above results, the query parameterization was on the selectivities of the base relations. Another parameter that is well-known to have significant impact on plan choices is the amount of system memory available for query processing (e.g. Nested Loop joins may be favored in low-memory environments, whereas Hash Joins may be a more attractive alternative in memory-rich situations). In fact, plan costs can be highly non-linear or even *discontinuous* at low memory availabilities [3, 4].

We found that the memory budget certainly had significant impact on the spatial layouts and cardinalities of the plan diagrams. For instance, with QT2 on OptA, the plan cardinality varied between 21 and 37 with varying memory for the buffer pages and the sort heap, as shown in Table 8. However, the basic observation that dense plan diagrams can be reduced to a few plans with low cost increase thresholds remained unchanged as shown in the last two columns of Table 8.

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

7.9 TPC-DS

We also validated our results on TPC-DS, the recently released decision support benchmark [27]. TPC-DS models the decision support functions of a retail product supplier, including data loading, multiple types of queries and data maintenance. The database consists of multiple snowflake schemas with shared dimension tables, skewed data and a large query set. We used a 100 GB sample database which has 24 tables, generated using the TPC-DS benchmark's synthetic generator, on OptC. Representative results are shown in Table 9 for sample query templates based on the TPC-DS queries. These plan diagrams were produced with 100 query points per dimension, uniformly distributed in the selectivity space. We see in the table that though these plan diagrams are dense, the plan reduction process produces reduced plan diagrams of low cardinality, seemingly *independent* of the properties and complexity of the underlying database.

Table 9: TPC-DS

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

8 Conclusions

In this report, we investigated from a variety of perspectives, the problem of reducing the dense plan diagrams produced by modern query optimizers, without adversely affecting the query processing quality. Our analysis shows that while finding the optimal reduction is NP-hard, the CostGreedy algorithm proposed here is able to efficiently provide a tight and optimal performance guarantee. Further, the experimental assessment on commercial optimizers indicates that in practice CostGreedy is always within a plan or two of the optimal, frequently giving the optimal itself. The AvgEst and AmmEst estimators are able to rapidly provide a fairly accurate assessment of the tradeoff between reduced plan cardinality and the cost threshold, helping users to focus on the interesting threshold ranges. Finally, the experimental study indicates that the graph of cardinality versus threshold is typically steep and that the number of plans in the reduced plan diagram is likely to be brought down to anorexic levels (within/around ten) with thresholds of around twenty percent even for high-dimensional query templates. These results are even more striking when we consider that they are *conservative* since a cost bounding rule was used, rather than the actual costs of replacement plans at query points.

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

A APPENDIX

A.1 PCM violation

For bounding the cost of a query point when it is to be replaced by a substitute plan, we assumed the Plan Cost Monotonicity (PCM) behaviour. While this is true for most of the query templates, for those that contain negation operators such as "set difference" or short-circuit operators like "exists", the PCM condition may not apply. In such cases, the query execution cost of a plan will be monotonically non-decreasing in another quadrant. For example, if there are negation operators for both the attributes in which selectivities vary and there is a reduction in the result cardinality, the cost function may be non-decreasing as we move in the third quadrant (i.e. it will be non-increasing with increase in the selectivity of input relations). In general, we assume that the cost behaviour is monotonic as we increase input selectivities. In such situations, to upper bound the cost of a query point q, we need to only consider the costs of all the query points in the appropriate quadrant of the plan diagram with q as the origin. Table 10 shows the quadrant that is to be considered for the possible cost behaviours in 2 dimensions. Thus, we only assume monotonicity in each dimension.

Table 10: Reduction Quadrants

Behaviour in X dimension	Behaviour in Y dimension	Dominating Quadrant
Non-decreasing	Non-decreasing	I
Non-increasing	Non-decreasing	II
Non-increasing	Non-increasing	III
Non-decreasing	Non-increasing	IV

A.2 Single-swallowing PlanRed

The Single-swallowing PlanRed problem is defined as follows:

Definition 6 *Single-swallowing PlanRed*

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

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

Applying the bounding rule of Section 4.1, the second condition is converted to the stronger requirement:

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

We find that enforcing the single-swallowing restriction does not change the complexity of the plan reduction problem. We show this by reducing a variation of the Dominating Set problem in a Directed Graph into an instance of Single-swallowing PlanRed.

For the purpose of our reduction, we will be using an instance of the Dominating Set problem where the directed acylic graph $G = (V, E)$ is connected and has the following structure

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

Lemma 7 *The Dominating Set problem in a Directed graph with the given structure is NP-Hard.*

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

1. For each $S_i \in S$, create a node v_i (v nodes) and for each element $e_i \in U$ create a node u_i (u nodes). Create another node w.
2. Let $V = \{u_1, u_2, \ldots u_m, v_1, v_2, \ldots v_n, w\}$
3. Let $E = \{(v_i, u_j) : e_j \in S_i\} \bigcup \{(w, v_i), \forall i = 1 \ldots n\}$

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

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

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

Theorem 6 *The Single-swallowing Plan Reduction Problem is NP-Hard.*

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

1. For each node v_i create a set $S_i = \{q_i\}$ and $U = U \bigcup \{q_i\}$
2. For each edge (v_i, v_j) perform $S_i = S_i \bigcup \{q_j\}$

ReducePlans($PlanDiagramP$**,** $threshold$**)**

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

Figure 14: Algorithm ReducePlans

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

This instance of the set cover problem can then be converted into a plan diagram by using the Algorithm Reduce given in Table 4. We make a slight modification in Algorithm Reduce, wherein, rather than choosing the set with smallest index as its representative color, we will instead choose the set with the same index as the element as its representative color. (This can be done because, while a set is created, a corresponding element is also created for it). We know by Lemma 3 that the optimal solution of the Plan Diagram formed by Algorithm Reduce gives the optimal solution of the Set Cover instance used as input to it. Also, this reduction runs in polynomial time. Hence, it will suffice for us to just show that the optimal solution to the plan diagram thus formed conforms with the aforementioned restriction.

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

A.3 Storage-budgeted PlanRed

Theorem 7 *The Storage-budgeted Plan Reduction Problem is NP-Hard.*

Proof: We prove the hardness of the problem by using it to solve the Plan Reduction problem. Assume that a polynomial time solution exists for the Storage-budgeted PlanRed problem. Let $FindPlans$ be polynomial time algorithm for the same. The algorithm $FindPlans$ takes as input the number of plans, and returns the plans chosen that minimizes the threshold, and let the method $findThreshold$ take as input these plans and return the threshold by which the cost of the query point increases.

Consider the algorithm ReducePlans given in Figure 14. Algorithm ReducePlans takes as input the threshold and returns the optimal number of plans that can replace the other plans without increasing the cost query points of beyond the given threshold.

1. Create a bucket B_i for each different plan i in P, and put all query points having the same plan in the corresponding bucket.
2. Create a border bucket BB_i for each different plan i in P. Using the Edge Detector algorithm, identify the border points of each contiguous plan region and only insert those points into the corresponding bucket.
3. Sort the buckets B_i in ascending order of the areas covered by their associated plans in P. Let this sorted list be $B_1, B_2, \ldots, B_n$
4. for $i = 1$ to n
   (a) Swallow (B_i) = true
   (b) for each point p in B_i
   (c) for $j = 1$ to n and $(j \neq i)$
      i. find, if available, a point q in BB_j such that q is in first quadrant w.r.t p, $cost(q)$ is within $[100\%, (100 + \lambda)\%]$ of $cost(p)$, and $cost(q)$ is the minimum across all such qualifying points in BB_j
   (d) if one or more q points are identified from the above step, choose the q point with the lowest $cost(q)$, and mark that point p can be assigned to qs bucket
   (e) else Swallow (B_i) = false
   (f) break
   (g) if Swallow(B_i)= true, move all the points in B_i to their assigned replacement buckets, then delete B_i and BB_i
5. Output all the points of P with their current plan assignments based on their assigned buckets, and use the associated coloring to form the reduced plan diagram R

Figure 15: Algorithm AreaGreedy

It can be seen that the algorithm runs in polynomial time (it calls the algorithm $FindPlans$ atmost n times where n is the number of plans in the Plan Diagram). Thus, we have a polynomial time solution to the PlanRed Problem if we have polynomial time solution to the Storage-budgeted PlanRed Problem, which means we have a polynomial time solution to the Set Cover problem. Hence the theorem. ■

A.4 AreaGreedy Algorithm

The detailed AreaGreedy Algorithm is given in Figure 15.

A.5 Efficiency of AreaGreedy

Lemma 8 *The approximation factor* $\frac{|AG|}{|Opt|} \geq 0.5\sqrt{m}$

Proof: Construct the plan diagram as follows.

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

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

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

It can be seen that

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

Hence, for this plan diagram ,

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

Hence the Lemma. ■

A.6 Performance bound of Algorithm ThresholdGreedy

Theorem 8 *Given a storage budget of k plans, let Ben_{opt} be the benefit obtained by the optimal solution's selection, and Ben_{TG} be the benefit obtained by the ThresholdGreedy algorithm's (Figure 16) selection. Then*

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

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

$$Ben_{TG} = \sum_{i=0}^{k} Ben_{P_i} \tag{4}$$

$$Ben_{opt} = \sum_{i=0}^{k} Ben_{Q_i} \tag{5}$$

ThresholdGreedy ($PlanDiagramP, Budgetk$**)**

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

Figure 16: Algorithm ThresholdGreedy

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

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

Since P_2 is chosen first, it can be seen that

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

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

Similarly for P_3 the following inequality can be formed.

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

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

In general these inequalities can be written as

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

Adding the above set of equations over all i and using (4) and (5) we obtain the following set of k inequalities.

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

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

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

$$...$$

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

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

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

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

Using the above two equations we get

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

■

References

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

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

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

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

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

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

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

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

[9] S. Ganguly and R. Krishnamurthy, "Parametric Query Optimization for Distributed Databases based on Load Conditions", *Proc. of COMAD Intl. Conf. on Management of Data*, December 1994.

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

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

[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] 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.

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

[16] 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.

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

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

[19] 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.

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

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

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

[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 VLDB Intl. Conf. on Very Large Data Bases (VLDB)*, September 2001.

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

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

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

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

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