On the Production of Anorexic Plan Diagrams

Harish D. Pooja N. Darera Jayant R. Haritsa*
Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore 560012, INDIA

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 TPCH-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

A query optimizer's execution plan choices, for a given database and system configuration, are primarily a function of the *selectivities* of the base relations in the query. In a recent paper [16], we introduced the concept of a "plan diagram" to denote a color-coded pictorial enumeration of the plan choices of the optimizer for a parameterized query template over the relational selectivity space. For example, consider QT8, the parameterized two-dimensional 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).

```
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 this picture[1], produced with the Picasso tool [15] on a commercial database engine, a set of 89 different optimal plans, P1 through P89, cover the selectivity space. The value associated with each plan in the legend indicates the percentage area covered by that plan in the diagram – P1, for example, covers about 22% of the space, whereas P89 is chosen in only 0.001% of the space.

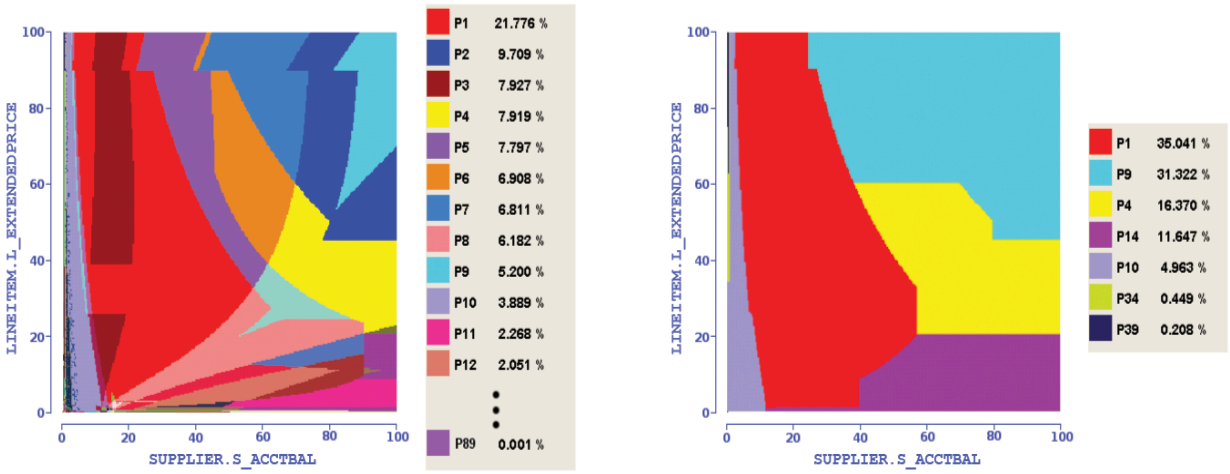
As is evident from Figure 2(a), plan diagrams can be extremely complex and dense, with a large number of plans covering the space – several such instances spanning a representative set of query templates based on the TPC-H benchmark, over a suite of industrial-strength optimizers, are available at [15]. However, it was also shown in [16] 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 users were willing to tolerate a minor cost increase of at most 10% for any query point in the diagram relative to its original (optimizer-estimated) cost, Figure 2(a) could be reduced to that shown in Figure 2(b), where only 7 plans remain – that is,

*Contact Author: haritsa@dsl.serc.iisc.ernet.in

VLDB '07, September 23-28, 2007, Vienna, Austria.

[1]The figures in this paper should ideally be viewed from a color copy, as the grayscale version may not clearly register the features.



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

Figure 2: Sample Plan Diagram and Reduced Plan Diagram (QT8)

most of the original plans have been "completely swallowed" by their siblings, leading to a highly reduced plan cardinality.

The complete graph of the reduced diagram's plan cardinality as a function of the cost increase threshold is shown in Figure 3.

Anorexic Plan Diagrams

In [16], it was concluded that "with a modest cost increase, two-thirds of the plans in a dense plan diagram are liable to be eliminated through plan swallowing". We make a stronger and significantly more impactful claim in this paper: A cost increase threshold of *only 20 percent* is usually amply sufficient to bring down the *absolute* number of plans in the final reduced picture to *within or around ten*. Further, that this applies not just to the 2D templates considered in [16], but also to *higher-dimensional* templates. In short, that plan diagrams can usually be made "anorexic" in an absolute sense while retaining acceptable query processing performance. This observation is based on our experience with a wide spectrum of dense plan diagrams ranging from tens to hundreds of plans, across the suite of industrial-strength optimizers, on TPC-H-based multi-dimensional query templates.

Carrying out anorexic plan reduction on dense plan diagrams has a variety of useful implications for improving both the efficiency of the optimizer and the choice of execution plan, as described in Section 2. Further, 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 made these choices.

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

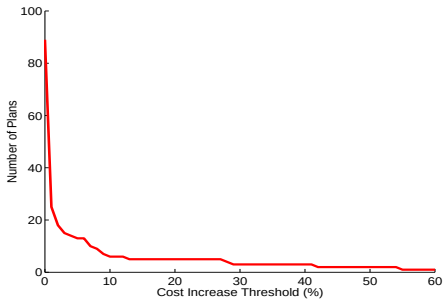


Figure 3: Plan Cardinality vs Cost Increase Threshold

more efficient than the $O(m^2)$ reduction algorithm of [16]. 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 Threshold-Greedy, 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-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 in this paper), has a variety of useful implications for improving both the efficiency of the optimizer and the choice of execution plan:

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, 7, 10, 11, 13]). 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. A practical difficulty with PQO, however, is the representation of the plan optimality boundaries, which could 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 suggested in [11]: For the current user query, 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 [11], the total time taken for many such costings may become comparable. But, 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 [12], 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 around (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 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 [14].

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.

Minimization of Multi-Plan Overheads: Multi-plan approaches that dynamically select the best plan at runtime by executing multiple different plans, either in parallel or sequentially, were proposed in [1, 12] – plan reduction can help to reduce the computational overheads of these approaches by minimizing the number of alternative choices.

3. RELATED WORK

To the best of our knowledge, apart from the initial results presented by us in [16], 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 [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 of 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*.

Subsequently, in [10, 11], a Volcano-style optimizer was modeled, and SPJ query templates with two, three and four-dimensional relational selectivities were evaluated. In their formulation, the cost

increase threshold cannot be guaranteed in the presence of non-linear 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 4D query template with 134 original plans is only reduced 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 for every individual query point, 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 [8]. 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 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 $\lambda \to 0$, R will be almost identical to P, whereas for $\lambda \to \infty$, R 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 $c_j(q)$. This means that the reductions discussed here are *conservative* since, 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 paper.

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 P is monotonically non-decreasing over the entire selectivity space S.

Intuitively, what the PCM condition states is that the query execution cost of a plan is expected 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. In this situation, the following rule applies:

***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. The reason for restricting attention to the first quadrant is that only a vacuous statement can be made about the costs of plans from other quadrants, namely that they lie in the interval $[c_1(q_1), \infty)$.

Moreover, 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. That is, condition 2 of Definition 2 is replaced by the stronger requirement

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

Handling non-PCM templates. When a query template features negation operators (e.g "set difference") or short-circuit operators (e.g. "exists"), the PCM condition may not hold. However, as long as the template exhibits monotonicity (non-decreasing or non-increasing) along each of the selectivity axes, the above Cost Bounding Rule still applies with an appropriate choice of reduction quadrant, as shown in Table 1 for the 2D case.

Table 1: Reduction Quadrants

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

In the remainder of the paper, we consider only the common situation of plan diagrams for which the PCM condition applies. Further, any plan diagram that has more than 10 plans is characterized as *dense*. We use n and m to denote the number of plans and the number of query points in P, respectively. The diagram resolutions in the X and Y axes are denoted by m_1 and m_2, 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 P.

4.2 The Set Cover Problem

The classical Set Cover problem is 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

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 smallest i such that query point $q \in S_i$ (i.e. denotes current plan of q 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 grid, 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 $(1 + \lambda)$.

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 $\lambda = 10\%$,

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 $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$; $1 \leq i_1 < i_2 < \cdots < i_k \leq n + 1$
   (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. $\forall z, z \in belong(q)$ s.t. $z > j$, 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} s.t. $r \in S_j$, 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

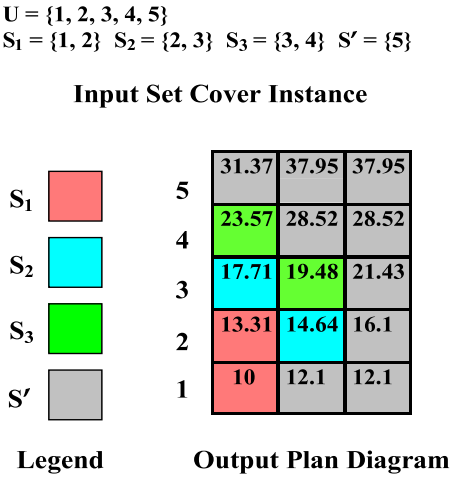


Figure 5: Example of Algorithm Reduce

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 grid (i.e. 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, i.e. 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 polynomial time complexity $O(nm)$.

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. □

As an aside, restricting the above problem to permit a plan to be swallowed only if it can be entirely replaced by a *single* sibling plan does not lower the problem complexity [9].

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.

This problem can be viewed as 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 [8] is used in [9] to prove the following theorem:

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

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.
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, use color that results in least cost increase.
11. End Algorithm CostGreedy

Figure 6: CostGreedy

5. GREEDY PLAN REDUCTION

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

We first consider AreaGreedy, the reduction algorithm proposed in [16], 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 PlanRed. We then present ThresholdGreedy, a greedy reduction that has strong performance bounds for the storage-budgeted variant. As before, for ease of exposition, we assume that P 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 [16] first sorts the plans featuring in P 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. A detailed description of the algorithm is available in [9].

By inspection, AreaGreedy clearly has a time complexity of $O(m^2)$, where m is the number of query points in P. With respect to reduction quality, let AG denote the solution obtained by AreaGreedy, and let Opt denote the optimal solution. Then, the upper bound of the approximation factor $\frac{|AG|}{|Opt|}$ is at least $0.5\sqrt{m}$ [9].

5.2 The CostGreedy Algorithm

We propose here 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 is 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 (under the λ constraint) with the plan corresponding to that set.

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

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

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 P.

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}. As 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 9).

In the above process, a swallowed point is recolored only *once*, in marked contrast to AreaGreedy where a swallowed point may be recolored multiple times before settling on its final color [9].

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 P.

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 ∞.

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 column $mincost[i]$ are candidates for the minimum values in column $i-1$, $mincost[i-1]$ is updated with the value $min(mincost[i], mincost[i-1])$.

With the above procedure, when moving to the next row to be processed, each $mincost[i]$ column 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 $2n$ processing iterations, 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 assess the approximation factor that can always be guaranteed by CostGreedy 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* P.

PROOF. It has been shown in [6, 19] 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,

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

□

Tightness of Bound. It is shown in [19] 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 [19] that the following tight bound of $\ln m$ for Set Cover can be achieved using such a construction when $m = G(k,l)$:

$$\ln m - \ln\ln m - 0.31 \leq \frac{k}{l} \leq \ln m - \ln\ln m + 0.78 \qquad (2)$$

These results are used in the following lemma.

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

PROOF. The construction process in [19] 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 employing the construction in [19]. Using this instance 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, yet 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

$$\frac{|Greedy|}{|Opt|} \geq \ln m - \ln\ln m - 0.31 \qquad (3)$$

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 variant (Section 4.4) of the PlanRed problem. Specifically, we present ThresholdGreedy, a greedy algorithm that selects plans based on maximizing the benefits obtained

ThresholdGreedy (PlanDiagram P, Budget k)

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 that 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 has multiple recoloring choices, use color resulting in least cost increase.
6. End Algorithm ThresholdGreedy

Figure 8: Algorithm ThresholdGreedy

by choosing them. The benefit of a plan is defined to be the extent to which it decreases the cost threshold λ of R 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 R is *a single plan* which corresponds to the plan of the TopRight query point in P. 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 P, i.e. $\lambda_{SinPlan} = cost(TopRight)/cost(BottomLeft)$.

We bootstrap the selection algorithm, shown in Figure 8, by first choosing this plan and subsequently choosing additional plans based on their relative benefits. 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 [9]):

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 $k \rightarrow \infty$, the ratio asymptotically goes down to 0.63. In an overall sense, this means that ThresholdGreedy is always guaranteed to provide close to *two-thirds of the optimal benefit*.

6. ESTIMATORS FOR PLAN REDUCTION

Our experience has been that CostGreedy takes only 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.

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.
6. return $|C|$
7. End Algorithm AvgEst

Figure 9: Algorithm AvgEst

In the above situations, using the CostGreedy method repeatedly to find the desired setting may prove to be 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 P. 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.

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 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 following experimental results.

7. EXPERIMENTAL RESULTS

Having considered the theoretical and statistical aspects of plan reduction in the previous sections, we now move on to presenting our experimental results. The testbed is the Picasso optimizer visualization tool [15], executing on a Sun Ultra 20 workstation with 4GHz processor, 4GB main memory and 240GB hard disk, running Windows XP Pro. Through the GUI of the Picasso tool, users can submit a query template, the grid resolution at which the instances of this template should be uniformly distributed across the selectivity space, the parameterized relations (axes) and their attributes on

Table 2: Computational Efficiency (QT8, Resolution=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

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 diagrams.

We conducted our plan reduction experiments over dense plan diagrams produced from a variety of multi-dimensional TPCH-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.

As mentioned above, we have experimented with a variety of query templates – however, due to space limitations, we present here the detailed results only for the sample two-dimensional QT8 query template described in the Introduction, and its higher-dimensional variants, on a representative optimizer. Summary results for other templates and optimizers, which are very similar in flavor, are also included.

7.1 Computational Efficiency

We start off by first quantitatively evaluating the runtimes of the two greedy algorithms, AreaGreedy (from [16]) and CostGreedy (proposed in this paper), as compared to the time taken to produce OptRed, the computationally-hard optimal solution. A sample set of results is shown in Table 2 for QT8 instantiated at a grid resolution of 100 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, OptRed 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 $n \ll m$ in practice (recall that n is the number of plans and m is the total number of query points in P).

7.2 Plan Reduction Quality

Turning our attention to the reduction quality, we see in Table 2 that AreaGreedy and CostGreedy are identical to OptRed, all three

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

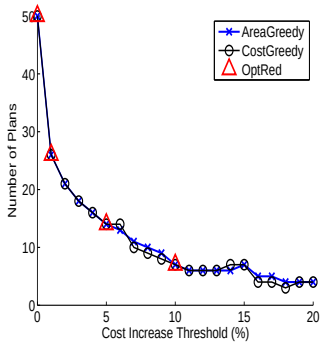


Figure 10: Reduction Quality (QT8, Res=100)

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 the Optimal due to its extremely high computational overheads).

Another point to note in Figure 10 is the initial steep exponential 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 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 Tables 3 and 4, which capture the reduction behavior of two popular commercial optimizers, Optimizer_A and Optimizer_B, with various TPC-H-based query templates on which they produced dense plan diagrams.

Further, while increasing the grid resolution may increase the number of plans in the original plan diagram (due to 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 means that for practical threshold settings, the final plan cardinality in the reduced diagram is essentially "scale-free" with regard to resolution.

Table 3: Optimizer_A (Res=100)

TPC-H Query Template	Original Plans	Reduced Plans (λ = 10%)	Reduced Plans (λ = 20%)
2	43	12	8
5	23	6	5
8	50	7	4
9	38	4	3
10	17	4	3

The above results were obtained with uniformly distributed data. When skewed data was used instead, the observed results did not materially change since data skew primarily affects the constants used to obtain a particular selectivity, but has comparatively little impact on the optimizer's selectivity-driven plan choices.

7.3 Scaling with Dimensions

The above results were obtained on a 2D query template, and we

Table 4: Optimizer_B (Res=100)

TPC-H Query Template	Original Plans	Reduced Plans (λ = 10%)	Reduced Plans (λ = 20%)
2	22	8	6
5	15	2	2
8	20	3	3
9	37	10	7
10	12	3	3

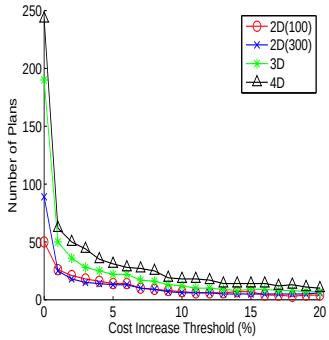


Figure 11: Scaling with Dimensions

now move to evaluating the effect of increased template dimensionality. Specifically, evaluating the behavior with 3D and 4D versions of the QT8 template (created through the addition of selectivity predicates on c_acctbal and o_totalprice).

The results are shown in Figure 11 for 2D with resolutions of 100 and 300 per dimension, 3D with resolution 100 per dimension, and 4D with resolution 30 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 5, the reduction behavior is remarkably stable: First, the location of the knee varies only marginally, occurring in the neighborhood of 10%. Second, the threshold required to bring the reduced plan cardinality down to 10 plans is within 20%, a very practical value from a user perspective, even in a 4D setting. Again, this seems to suggest that for practical threshold settings, the final plan cardinality in the reduced diagram is essentially "scale-free" with regard to dimension.

7.4 Estimator Performance

Our next experiment studies the quality of the *knee estimates* provided by the estimators. The results are shown in Figure 12 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 quantitatively captured in Table 6.

Table 5: 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%

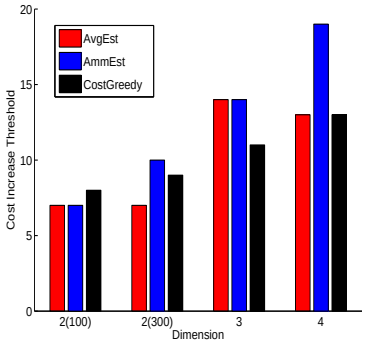


Figure 12: Knee Estimates

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 6: 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

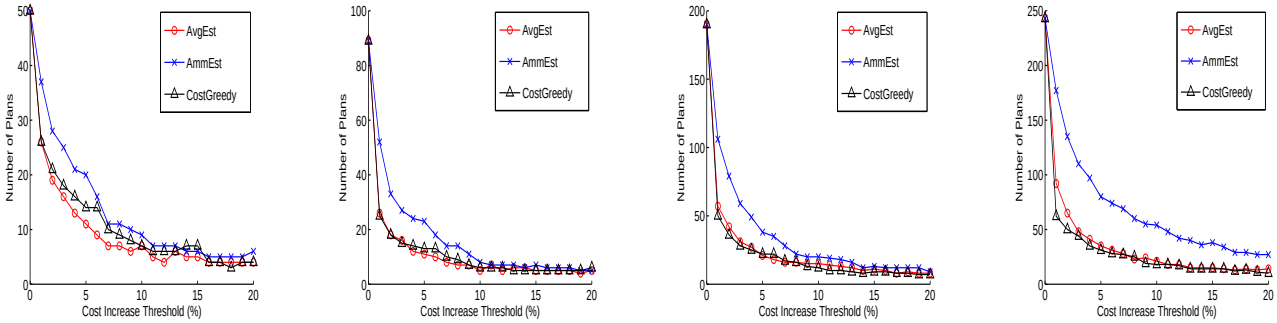
7.5 Effect of Memory Availability

In all the above results, the query parameterization was on the selectivities of the base relations. Another parameter that is well-known to have significant impact on plan choices is the amount of system memory available for query processing (e.g. Nested Loops 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 conducted experiments wherein the memory was varied from the full system memory to the minimum permitted by the engine (in our setup, this corresponded to going from 4GB to 16MB). We found that the memory budget certainly had significant impact on the spatial layouts and cardinalities of the plan diagrams – for example, the plan diagram cardinalities went up significantly with decreased memory. However, the basic observation that dense plan diagrams can be reduced to a few plans with low cost increase thresholds remained unchanged. In short, while memory has a significant impact on the *optimization* process, it does not seem to materially affect the *reduction* process.

8. CONCLUSIONS

In this paper, 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



(a) **Est-2D (100)** (b) **Est-2D (300)** (c) **Est-3D** (d) **Est-4D**

Figure 13: Estimator Performance

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. In our future work, we plan to extend the analysis to the richer query templates of the recently announced TPC-DS benchmark [22].

Acknowledgments. This work was supported in part by a Swarnajayanti Fellowship from the Dept. of Science & Technology, Govt. of India. We thank the reviewers for their valuable comments.

9. REFERENCES

[1] G. Antonshenkov, "Dynamic Query Optimization in Rdb/VMS", *Proc. of 9th IEEE Intl. Conf. on Data Engineering (ICDE)*, April 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 Symp. on Principles of Database Systems (PODS)*, May 1999.

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

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

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

[9] Harish D., P. Darera and J. Haritsa, "Reduction of Query Optimizer Plan Diagrams", *Tech. Rep. TR-2007-01, DSL/SERC, Indian Inst. of Science*, 2007. *http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2007-01.pdf*

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

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

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

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

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