Efficiently Approximating Query Optimizer Plan Diagrams

Atreyee Dey Sourjya Bhaumik Harish D. Jayant R. Haritsa*
Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore 560012, INDIA

ABSTRACT

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

In this paper, we investigate strategies for efficiently producing close approximations to complex plan diagrams. Our techniques are customized to the features available in the optimizer's API, ranging from the generic optimizers that provide only the optimal plan for a query, to those that also support costing of sub-optimal plans and enumerating rank-ordered lists of plans. The techniques collectively feature both random and grid sampling, as well as inference techniques based on nearest-neighbor classifiers, parametric query optimization and plan cost monotonicity.

Extensive experimentation with a representative set of TPC-H and TPC-DS-based query templates on industrial-strength optimizers indicates that our techniques are capable of delivering 90% accurate diagrams while incurring less than 15% of the computational overheads of the exhaustive approach. In fact, for full-featured optimizers, we can guarantee zero error with less than 10% overheads. These approximation techniques have been implemented in the publicly available Picasso optimizer visualization tool.

1. INTRODUCTION

For a given database and system configuration, a query optimizer's execution plan choices are primarily a function of the *selectivities* of the base relations in the query. In a recent paper [20], 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.

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

VLDB '08, August 24-30, 2008, Auckland, New Zealand

For example, consider QT8, the parameterized 2D query template shown in Figure 1, based on Query 8 of the TPC-H benchmark [28]. Here, selectivity variations on the SUPPLIER and LINEITEM relations are specified through the s_acctbal :varies and l_ extendedprice :varies predicates, respectively. The associated plan diagram for QT8 is shown in Figure 2(a), produced with the Picasso optimizer visualization tool [23] on a popular commercial database engine.

```
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
```

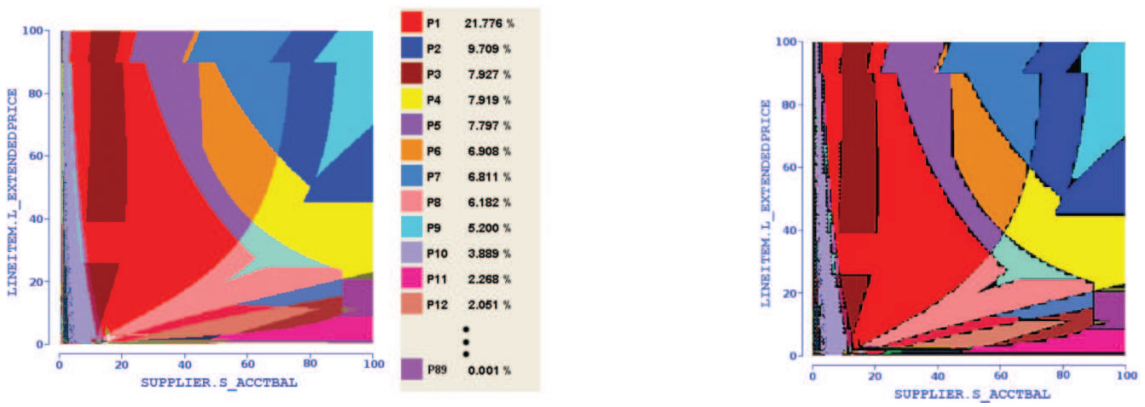
Figure 1: Example Query Template QT8

In this picture[1], each colored region represents a specific plan, and a set of 89 different optimal plans, P1 through P89, cover the selectivity space. The value associated with each plan in the legend indicates the percentage area covered by that plan in the diagram – the biggest, P1, for example, covers about 22% of the space, whereas the smallest, P89, is chosen in only 0.001% of the space.

Applications of Plan Diagrams

Since their introduction in [20] a few years ago, plan diagrams have proved to be a powerful metaphor for the analysis and redesign of industrial-strength database query optimizers. For example, as evident from Figure 2(a), they can be surprisingly complex and dense, with a large number of plans covering the space – several such instances spanning a representative set of benchmark-based query templates on current optimizers are available at [23]. Our interactions with industrial development teams have indicated that these

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



(a) **Plan Diagram** (b) **Approximate Diagram (10% Error Bound)**

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

diagrams have often proved to be contrary to the prevailing conventional wisdom. The reason is that while optimizer behavior on *individual queries* has certainly been analyzed extensively by developers, plan diagrams provide a completely different perspective of behavior *over an entire space*, vividly capturing plan transition boundaries and optimality geometries. So, in a literal sense, they deliver the "big picture".

Plan diagrams are currently in use at various industrial and academic sites for a diverse set of applications including analysis of existing optimizer designs; visually carrying out optimizer regression testing; debugging new query processing features; comparing the behavior between successive optimizer versions; investigating the structural differences between neighboring plans in the space; evaluating the variations in the plan choices made by competing optimizers; etc. As a case in point, visual examples of non-monotonic cost behavior in commercial optimizers, indicative of modeling errors, were highlighted in [20].

Apart from optimizer design support, plan diagrams can also be used in operational settings. Specifically, since they identify the optimal set of plans for the entire relational selectivity space at compile-time, they can be used at run-time to immediately identify the best plan for the current query without going through the time-consuming optimization exercise. Further, they can prove useful to adaptive plan selection techniques (e.g. [5, 6, 17]) which, based on the differences between the actual selectivities encountered during execution and the associated compile-time estimates, may dynamically choose to reoptimize the query and switch plans mid-way through the processing. In this context, plan diagrams can help to eliminate the reoptimization overheads incurred in determining the substitute plan choices for estimation errors that occur on the selectivity dimensions.

Plan Diagram Reduction. A particularly compelling utility of plan diagrams is that they provide the input to "plan diagram reduction" algorithms. Specifically, given a plan diagram and a cost-increase-threshold (λ) specified by the user, these reduction algorithms recolor the dense diagram to a simpler picture that features only a subset of the original plans while ensuring that the cost of no individual query point goes up by more than λ percent, relative to its original cost. That is, some of the original plans are "completely swallowed" by their siblings, leading to a reduced plan cardinality in the diagram. It has been shown last year [12] that if users were willing to tolerate a minor cost increase of λ =20%, the absolute number of plans in the final reduced picture could be brought down to *within or around ten*. In short, that complex plan diagrams can be made "anorexic" while retaining acceptable query processing performance. For example, the reduced version of the QT8 plan diagram (Figure 2(a)) retains *only 5* of the original 89 plans with λ =20%.

Anorexic plan diagram reduction has significant practical benefits [12], including quantifying the redundancy in the plan search space, enhancing the applicability of parametric query optimization (PQO) techniques [14, 15], identifying error-resistant and least-expected-cost plans [3, 4], and minimizing the overhead of multi-plan approaches [1, 16]. A detailed study of its application to identifying robust plans that are resistant to errors in relational selectivity estimates is available in [13].

Generation of Plan Diagrams

The generation and analysis of plan diagrams has been facilitated by our development of the Picasso optimizer visualization tool [23]. Given a multi-dimensional SQL query template like QT8 and a choice of database engine, the Picasso tool automatically produces the associated plan diagram. It is operational on several major platforms including IBM DB2, Oracle, Microsoft SQL Server, Sybase ASE and PostgreSQL. The tool, which is freely downloadable, is now in use by the development groups of several major database vendors, as also by leading industrial and academic research labs worldwide.

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

The above exhaustive approach is eminently acceptable for diagrams on low-dimension (1D and 2D) query templates with coarse resolutions (upto 100 points per dimension). However, it becomes impractically expensive for higher dimensions and fine-grained resolutions due to the exponential growth in overheads. For example, a 2D plan diagram with a resolution of 1000 on each selectivity dimension, or a 3D plan diagram with a resolution of 100 on each dimension, both require invoking the optimizer a *million* times. Even with a conservative estimate of about half-second per optimization, the total time required to produce the picture is close to a week! Therefore, although plan diagrams have proved to be extremely useful, their high-dimension and/or fine-resolution versions pose serious computational challenges.

Two obvious mechanisms to lower the computational time overheads are: (a) Customize the resolution on each dimension to be domain-specific – for example, coarse resolutions may prove sufficient for categorical data; and (b) Use computational units in parallel to leverage the independence between the optimizations of the individual query points, resulting in concurrent issue of multiple optimization requests.

In this paper, we consider how we can supplement the above remedies, which may not always be applicable or feasible, through the use of generic *algorithmic* techniques, as described next.

Approximate Plan Diagrams

Specifically, we investigate whether it is possible to efficiently produce *accurate approximations* to plan diagrams. Denoting the true plan diagram as P and the approximation as A, there are two categories of errors that arise in this process:

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

The ϵ_I error is challenging to control since a majority of the plans appearing in plan diagrams, as seen in Figure 2(a), are very small in area, and therefore hard to find.

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

The ϵ_L error is also challenging to control since the plan boundaries, as seen in Figure 2(a), can be highly non-linear, and are sometimes even irregular in shape [23].

Optimizer Classes

Our study shows that the ability to reduce overheads is a function of the plan-related functionalities offered by the optimizer's API, based on which we define the following three categories of optimizers:

Class I: OP Optimizers This class refers to the generic cost-based optimizers that are routinely found in virtually every enterprise database product, where the API only provides the optimal plan (OP), as determined by the optimizer, for a user query.

Class II: OP + FPC Optimizers This class of optimizers additionally provide a "foreign plan costing" (FPC) feature in their API, that is, of costing plans *outside* their native optimality regions. Specifically, the feature supports the following "what-if" question: "What is the estimated cost of sub-optimal plan p if utilized at query location q?". FPC has become available in the current versions of several industrial-strength optimizers, including DB2 [24] (Optimization Profile), SQL Server [25] (XML Plan), and Sybase [26] (Abstract Plan).

Class III: OP + FPC + PRL Optimizers This class of optimizers support, in addition to FPC, an API that provides not just the best plan but a "plan-rank-list" (PRL), enumerating the top k plans for the query. For example, with $k = 2$, both the best plan and the second-best plan are obtained when the optimizer is invoked on a query. Note that the PRL feature can be easily implemented in current optimizers through minor changes in the Dynamic Programming-based query optimization process – the details are given in [7]. However, to our knowledge, it is not yet available in any of the current systems. Therefore, we showcase its utility through our own implementation in a public-domain optimizer.

Approximation Techniques and Results

For Class I (OP) and Class II (OP+FPC) optimizers, the techniques that we propose are based on a combination of sampling and inference, while for Class III optimizers (OP+FPC+PRL), it is purely based on inference. The sampling techniques include both classical random sampling and grid sampling, while the inference approaches rely on nearest-neighbor (NN) classifiers [22], parametric query optimization (PQO) [14, 15] and plan cost monotonicity (PCM) [12]. For some of the techniques, theoretical results that help to provide guaranteed bounds on the errors are available, whereas for the others, empirical evaluation is the only recourse.

We have quantitatively assessed the efficacy of the various strategies, with regard to plan identity and location errors, through extensive experimentation with a representative suite of multi-dimensional TPC-H and TPC-DS-based query templates on leading commercial and public-domain optimizers. Our results are very promising since they indicate that accurate approximations can indeed be obtained *cheaply and consistently*, as described below.

10 percent Error Bound. Consider the case where the user desires that the approximation error is of the order of *10 percent* or less on both plan identities and plan locations. For Class I (OP) optimizers, it is possible to regularly achieve this target with *only around 15% overheads* of the brute-force exhaustive method. To put this in perspective, the earlier-mentioned one-week plan diagram can be produced in less than a day. A sample approximate diagram (having 10% identity and 10% location error) is shown in Figure 2(b), with all the erroneous locations marked in black – as can be seen, the approximation is materially faithful to the features of the true plan diagram, with the errors thinly spread across the picture and largely confined to the plan transition boundaries.

For Class II (OP+FPC) systems, a similar error performance is achieved with *only around 10% overheads*. An important point to note here is that plan costing is considerably cheaper than searching for the optimal plan. Finally, for Class III (OP+FPC+PRL) systems, the overheads come down to *less than 5%*.

1 percent Error Bound. We have also investigated the scenario where the user has the extremely stringent expectation of around *1 percent* plan identity and location errors. For this situation, Class I and II both take upto *40% overheads*, while Class III usually incurs less than *10% overheads*.

Contributions

In a nutshell, we present in this paper a range of techniques, customized to the optimizer's API richness, for efficiently generating

accurate approximate plan diagrams. These results are summarized in Table 1, where the typical range of overheads (relative to the exhaustive approach) is shown as a function of the user's error bound for each optimizer class.

Optimizer Class	Overheads Range (Bound = 10%)	Overheads Range (Bound = 1%)
Class I (OP)	1% – 15%	15% – 40%
Class II (OP+FPC)	1% – 10%	15% – 40%
Class III (OP+FPC+PRL)	1% –5%	1% – 10%

Table 1: Summary Results

Cost Diagrams. While not mentioned earlier, our techniques can be extended to produce the "cost diagram" [20] associated with each plan diagram. The cost diagram is a visualization of the estimated plan execution costs over the relational selectivity space. For Class I optimizers, an approximate cost diagram is generated through interpolation, while for Class II and Class III optimizers the exact cost diagram corresponding to the approximate plan diagram is obtained through the FPC feature. The complete details are given in [7].

Organization

The remainder of this paper is organized as follows: The approximation algorithms are presented in Section 2. Our experimental framework and performance results are highlighted in Section 3. Finally, in Section 4, we summarize our conclusions and outline future research avenues.

2. APPROXIMATION ALGORITHMS

In this section, we describe our suite of strategies for the efficient generation of approximate plan diagrams. We begin with algorithms for Class I optimizers, and then describe how these techniques can be improved for Class II optimizers leveraging their foreign-plan-costing (FPC) feature. We conclude with two variants of an algorithm for Class III optimizers with FPC and plan-rank-list (PRL) functionalities – the first version *guarantees zero error*, while the second trades error for further reduction in computational overheads.

For ease of presentation, we will assume in the following discussion that the query template is 2D – the extension to n-dimensions is straightforward and given in [7]. The true plan diagram is denoted by P and the approximation as A, with the total number of query points in the diagrams denoted by m. Each query point is denoted by $q(x, y)$, corresponding to a unique query with selectivities x, y in the X and Y dimensions, respectively. The terms $p_P(q)$ and $p_A(q)$ are used to refer to the plans assigned to query point q in the P and A plan diagrams, respectively (when the context is clear, we drop the diagram subscript).

Finally, the plan identity and plan location errors of an approximate diagram are defined as

$$\epsilon_I = \frac{|P| - |A|}{|P|} * 100 \quad \text{and} \quad \epsilon_L = \frac{|p_A(q) \neq p_P(q)|}{m} * 100,$$

respectively. The approximation techniques should ideally ensure that ϵ_I and ϵ_L are within the user-specified bounds θ_I and θ_L, or are at least in their close proximity. For simplicity of exposition, we will assume in the sequel that users specify the same bound $\theta_I = \theta_L = \theta$ on both metrics.

2.1 Class I Optimizers

The approximation procedures for this class of optimizers operate in two steps:

Optimization Step: In this step, a set of query points in the plan diagram are explicitly optimized to obtain the optimal plan choices at those points.

Inference Step: In this step, the plan choices for a set of unoptimized points are *inferred* using the results from the Optimization step.

For the random sampling-based algorithms, the above two steps are sequential, whereas for the grid-sampling-based algorithms, the steps are interleaved.

2.1.1 Random Sampling with NN Inference (RS_NN)

In the RS_NN algorithm, we first use the classical random sampling (without replacement) technique to sample query points from the plan diagram that are to be optimized during the optimization step. Since we have empirically found that with this technique, the plan-identity error ϵ_I is almost always greater than the plan-location error ϵ_L, the stopping criterion for the sampling is based on the former metric. The problem of finding the distinct plans in the plan diagram can be related to the classical statistical problem of finding distinct classes in a population [10]. Applying the recent results of [2], we obtain the following: Let s samples be taken on the plan diagram, let d_s be the number of distinct plans in these samples, and let f_1 denote the number of plans occurring only *once* in the samples. Then, it is highly likely that the number of distinct plans, d, in the entire plan diagram is in the cardinality range $[d_s, d_{max}]$, where

$$d_{max} = (\frac{m}{s} - 1)f_1 + d_s \tag{1}$$

If we ensure that the sampling is iteratively continued until d_s is within ϵ_I of d_{max}, then it is highly likely that the number of plans found thus far in the sample is within ϵ_I of d. Therefore, the RS_NN algorithm continuously evaluates Equation 1 to determine when the optimization step can be terminated.

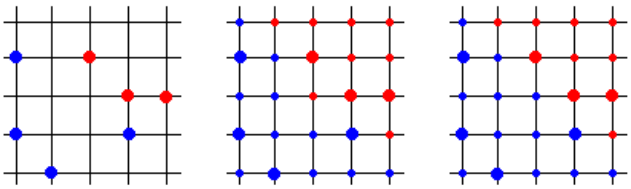
Our experience, as borne out by the experimental results given in [7], has been that the above stopping condition may be too conservative in that it takes many more samples than is strictly necessary. Therefore to refine the stopping condition, we use $\hat{d}_{ML}$, the most-likely-value estimator for d, defined in [2] as

$$\hat{d}_{ML} = (\sqrt{\frac{m}{s}} - 1)f_1 + d_s \tag{2}$$

which has an expected ratio error bound of $O(\sqrt{\frac{m}{s}})$.

We now terminate the optimization phase in two steps as follows: After d_s increases to a value within a $(1 - \delta)$ factor of d_{max}, we continue the sampling until d_s reaches to within a $(1 - \theta)$ factor of $\hat{d}_{ML}$. The value of δ conducive to good performance results has been empirically determined to be 0.3. The intuition behind this method is that once the gap between d_s and d_{max} has narrowed to a sufficiently small range, then the most-likely-value estimator can be used as a reliable indicator of the plan cardinality in the diagram.

Inference. After the completion of the sampling phase, the plan choices at the unoptimized points of the plan diagram need to be inferred from the plan choices made at the sampled points. This is done using a Nearest Neighbor (NN) style classification scheme [22]. Specifically, for each unoptimized point q_u, we search for the nearest optimized point q_o, and assign the plan of q_o to q_u. If there are multiple nearest optimized points, then the plan that occurs most often in this set is chosen, and in case of a tie on this metric, a random selection is made from the contenders.



(a) **Optimize** (b) **Infer** (c) **Filter**

Figure 3: Execution Stages of the RS_NN Algorithm

The distance between two query points $q_1(x_1, y_1)$ and $q_2(x_2, y_2)$ can be calculated using various distance metrics. We have evaluated the following three popular metrics: (1) *Manhattan* (L_1 norm); (2) *Euclidean* (L_2 norm); and (3) *Chessboard* (L_∞ norm). Our experience has been that the Chessboard Distance is most suitable, since the transition boundaries between plans often tend to be aligned along the (horizontal and vertical) axes. The same metric is also used for establishing the geometries of plan clusters in PLASTIC [8, 21], a tool designed to amortize query optimizer overheads.

Low Pass Filter (LPF). Inference using the NN scheme is well-known to result in boundary errors [22] – in our case, along the plan optimality boundaries. To reduce the impact of this problem, we apply a low-pass filter [9] after the initial inference has assigned plans to all the points in the diagram. The filter operates as follows: For each unoptimized point q_u, all its neighbors (both optimized and unoptimized) at a distance of one, are examined to find the plan that is assigned to more than half of the neighbors. If such a plan exists, it is assigned to q_u, otherwise the original assignment is retained.

Note that each unoptimized point is processed once during the filter phase, and that the resultant diagram is dependent on the order in which the points are taken up for processing. Further, the filter could, in principle, be applied multiple times. However, our empirical results indicate that the choice of processing order has only minuscule impact on overall diagram accuracy – in our implementation, the points are processed starting from the top right corner and moving towards the origin in reverse row-major order. Also, applying the filter multiple times does not provide any perceptible improvement – therefore, we apply it only once.

The functioning of the RS_NN algorithm is illustrated in Figure 3 – in this set of pictures, each large dot indicates an optimized query point, whereas each small dot indicates an inferred query point. The initial set of optimized sample query points is shown in Figure 3(a), and the NN-based inference for the remaining points is shown in Figure 3(b). Applying the LPF filter results in Figure 3(c) – note that the center query point, which has an (inferred) red plan in Figure 3(b), is re-assigned to the blue plan in Figure 3(c).

The complete RS_NN algorithm is shown in Figure 4. In our implementation, the initial number of samples s_0 is set to 1% of the space, and the increment in the number of samples after each iteration is also set to this value.

2.1.2 *Grid Sampling with PQO Inference (GS_PQO)*

We now turn our attention to an alternative approach based on *grid sampling*. Here, a low resolution grid of the plan diagram is first formed, which partitions the selectivity space into a set of smaller rectangles. The query points corresponding to the corners of all these rectangles are optimized first. Subsequently, these points are used as the seeds to determine which of the other points

RS_NN (QueryTemplate QT, ErrorBound θ, InitSamples s_0)

1. stage = 1; $s = s_0$;
2. Optimize s samples chosen uniformly at random.
3. Compute the values of d_{max} and $\hat{d}_{ML}$
4. if (stage = 1) then
5. if $d_s \geq ((1-\delta)d_{max})$ then stage = 2;
6. if (stage = 2) then
7. if $d_s \geq ((1-\theta)\hat{d}_{ML})$ then Go to Step 9.
8. Go to Step 2.

9. for each unoptimized point q_u
10. Determine the set N of nearest optimized neighbors.
11. Determine plan p which occurs most often in N.
12. In case of a tie set p by picking any plan at random from the contenders.
13. Assign the plan p to q_u.

14. for each unoptimized point q_u
15. Determine the set N' of neighbors at distance 1
16. if a plan p' occupies majority of N'
17. Overwrite the plan at q_u with p'.
18. End Algorithm RS_NN

Figure 4: The RS_NN Algorithm

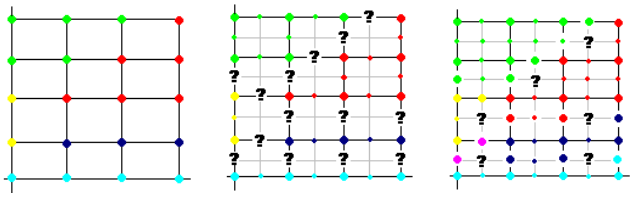
in the diagram are to be optimized.

Specifically, if the plans assigned to the two corners of an edge of a rectangle are the same, then the mid-point along that edge is also assigned the same plan. This is essentially a specific inference based on the guiding principle of the Parametric Query Optimization (PQO) literature (e.g. [14]): "If a pair of points in the selectivity space have the same optimal plan p_i, then all locations along the straight line joining these two points will also have p_i as their optimal plan." At first glance, our usage of the PQO principle here may seem at odds with our earlier observation in [20] that, for industrial-strength optimizers, this principle is observed more in the breach than in the observance. However, the difference is that we are applying PQO at a "micro-level", that is, within the confines of a small rectangle in the selectivity space, whereas earlier work has effectively considered PQO as a universal truth that holds across the entire space. Our experimental experience has been that micro-PQO generally holds in all the plan diagrams that we have analyzed.

When the plans assigned to the end points of an edge are different, then the midpoint of this edge is optimized. After all sides of a given rectangle are processed, its center-point is then processed by considering the plans lying along the "cross-hair" lines connecting the center-point to the mid-points of the four sides of the rectangle. If the two end-points on one of the cross-hairs match, then the center-point is assigned the same plan (if both cross-hairs have matching end-points, then one of the plans is chosen randomly). If none of the cross-hairs has matching endpoints, the center-point is optimized. Now, using the cross-hairs, the rectangle is divided into four smaller rectangles, and the process recursively continues, until all points in the plan diagram have been assigned plans.

The progress of the GS_PQO algorithm is illustrated in Figure 5 (again, each large dot indicates an optimized point, whereas each small dot indicates an inferred point). Figure 5(a) shows the state after the initial grid sampling is completed. Then, the '?' symbols in Figure 5(b) denote the set of points that are to be optimized in the following iteration as we process the sides of the rectangles. Finally, Figure 5(c) enumerates the set of points that are to be optimized while processing the cross-hairs.

A limitation of the GS_PQO algorithm is that it may perform a substantial number of unnecessary optimizations, especially when



(a) **Initial Grid** (b) **Mid Points** (c) **Cross Hairs**

Figure 5: Execution Loop in GS_PQO Algorithm

a rectangle with different plans at its endpoints features only a small number of new plans within its enclosed region. This is because GS_PQO does not distinguish between sparse and dense low-resolution rectangles. For example, if two similar-sized rectangles each have two plans featured at their four corner points, then they are divided similarly irrespective of the expected number of new plans present in the interior. We attempt to address this issue by refining the algorithm in the following manner: Assign each rectangle R with a "plan-richness" indicator $\rho(R)$ that serves to characterize the expected plan density in R, and then preferentially assign optimizations to the rectangles with higher ρ.

Our strategy to assign ρ values is as follows: Instead of merely making a *boolean* comparison at the corners of the rectangle as to whether the plans at these points are identical or not, we now dig deeper and compare the *plan operator trees* associated with these plans in order to estimate interior plan density. As an extreme example, consider the case where there is a left-deep tree at one corner of the rectangle, and a right-deep tree at another corner. In this situation, it seems reasonable to expect that there will be a significant number of plans in the interior of the rectangle since the process of shifting from a left-deep to a right-deep tree usually occurs in incremental intermediate steps, each corresponding to a new plan, rather than all at once – we have confirmed this observation through detailed analysis of the plan diagrams of industrial optimizers.

Plan Tree Differencing. Let the operator trees corresponding to a pair of plans p_i and p_j be denoted by T_i and T_j, respectively. Our comparison strategy is based on identifying and mapping similar operator nodes in the two trees. Figure 6 shows an example pair of plan trees T_1 and T_{10} corresponding to the plans p_1 and p_{10} that feature in the plan diagram of Figure 2(a) – the white nodes depict matching nodes, whereas the colored nodes represent distinct nodes.

In the following description, the term *branch* is used to refer to any directed chain of unary-input nodes between a pair of binary-input nodes, or between a binary node and a leaf, in these trees. Branches are directed from the lower node to the higher node, modeling the direction of data propagation.

The matching proceeds as follows:

1. First, all the leaf nodes (relations) and all the binary-input nodes (typically join nodes) are identified for T_i and T_j.

2. A leaf of T_i is matched with a leaf of T_j if and only if they both have the same relation name. In the situation that there are multiple matches available (that is, if the same relation name appears in multiple leaves), an edit-distance computation is made between the branches of all pairs of matching leaves between T_i and T_j. The assignments are then made in increasing order of edit-distances. For example, the NATION node appears twice in T_1 and T_{10} of Figure 6, and the specific pairing is made based on the closeness of matching in the branches arising out of these nodes.

3. A binary node of T_i is matched with a binary node of T_j if the set of base relations that are processed is the same. If the node operator names and the left and right inputs are identical (in terms of base relations), the nodes are made white. However, if the node operator names are different, or if the left and right input relation subsets are different, then the nodes are colored.

4. A minimal edit-distance computation is made between the branches arising out of each pair of matched nodes, and the nodes that have to be added or deleted, if any, in order to make the branches identical, are colored. Unmodified nodes, on the other hand, are matched with their counterparts in the sibling tree and made white.

5. Finally, each pair of matched nodes is assigned the same unique number in both trees. For example, the number 9 is assigned to the final join node, representing the composite relation formed by the join of all the base relations, in each tree.

Plan Richness Metric. We now describe the procedure to quantify plan richness in terms of plan-tree differences. Our formulation uses $|T_i|$ and $|T_j|$ to represent the number of nodes in plan-trees T_i and T_j, respectively, and $|T_i \cap T_j|$ to denote the number of matching nodes between the trees.

Now, ρ is measured as the classical Jaccard Distance [22] between the trees of the two plans, and is computed as

$$\rho(T_i, T_j) = 1 - \frac{|T_i \cap T_j|}{|T_i \cup T_j|} \tag{3}$$

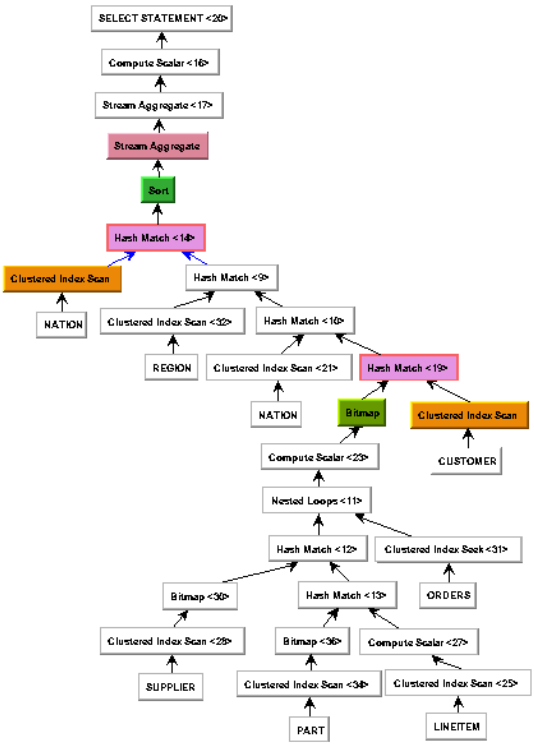
For example, the ρ for the plan tree pair (T_1, T_{10}) in Figure 6 is $1 - \frac{26}{38} = 0.32$.

While Equation 3 works for a pair of plans, we need to be able to extend the metric to handle an arbitrary set of plans, corresponding to the corners of the hyper-rectangle in the selectivity space. Given a set of n trees $\{T_1, T_2, \ldots, T_n\}$, this is achieved through the following computation:

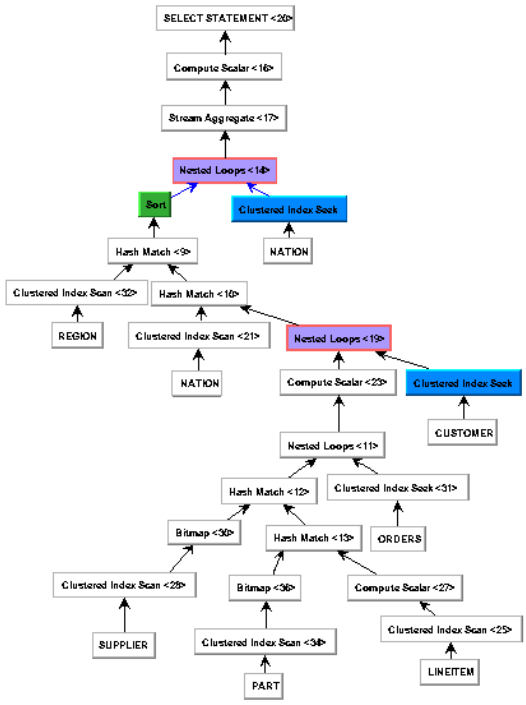
$$\rho(T_1, \ldots, T_n) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \rho(T_i, T_j)}{\binom{n}{2}} \tag{4}$$

Note that the ρ values are normalized between 0 and 1, with values close to 0 indicating that all the plans are structurally very similar to each other, and values close to 1 indicating that the plans are extremely dissimilar. Figure 7 depicts the ρ values calculated for a sample plan diagram (produced from a query template based on TPC-H Query 9) after partitioning into 20x20 squares. We see here that ρ reaches high values close to the origin and along the selectivity axes. This is in accordance with earlier observations in [14, 15, 18, 19, 20] that plans tend to be densely packed in precisely these regions of the selectivity space.

We now describe how GS_PQO utilizes the above characterization of plan-tree-differences. First, the grid sampling procedure is executed as mentioned earlier. Then, for each resulting rectangle, the ρ value is computed based on the plan-trees at the four corners, using Equation 4. The rectangles are organized in a max-Heap structure based on the ρ values, and the optimizations are directed towards the rectangle R_{top} at the top of the heap, i.e. with the current highest value of ρ. Specifically, the PQO principle is applied to the mid-points of all qualifying edges (those with common plans at both ends of the edge) in R_{top}, and all the remaining



(a) Plan Tree T_1



(b) Plan Tree T_{10}

Figure 6: Example of Plan Tree Difference

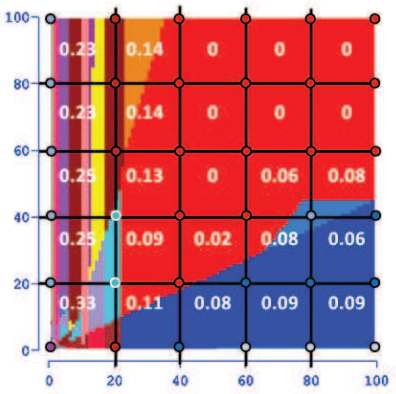


Figure 7: ρ values in Plan Diagram

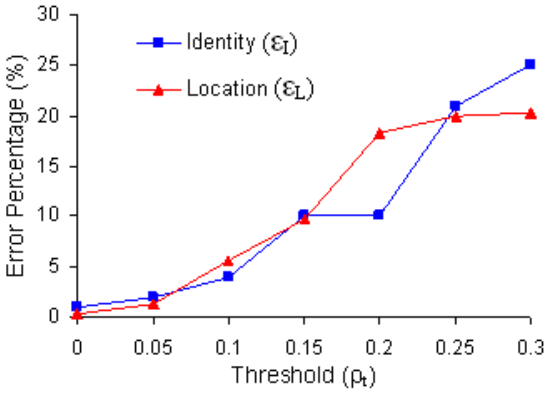


Figure 8: Effect of ρ_t on ϵ_I and ϵ_L Errors

edge mid-points are explicitly optimized. The rectangle is then split into four smaller rectangles, for whom the ρ values are recomputed, and these new rectangles are then inserted into the heap.This process continues until all the rectangles in the heap have a ρ value that is below a threshold ρ_t. The threshold is a function of the θ bound given by the user, with lower thresholds corresponding to tighter bounds.

A representative behavior of the error metrics, ϵ_I and ϵ_L, as a function of the ρ_t threshold, is shown in Figure 8, obtained with query template QT8. Note that unlike in the random sampling approach, we find here that ϵ_L does not always lag ϵ_I. The reason is that samples cease to be assigned to rectangles with $\rho \leq \rho_t$ even when they contain more than one plan. In this situation, inference may increase ϵ_L due to erroneous boundary detection between the plans present inside the rectangle. Therefore, even when ϵ_I is low, ϵ_L may have a comparatively larger value.

However, we also see in Figure 8 that setting the threshold equal to the error bound, i.e. $\rho_t = \theta$ (e.g. for $\theta = 10\%, \rho_t = 0.1$), is an adequate heuristic that is sufficient to meet user expectations on both error metrics – we observed similar behavior for other query templates as well, and therefore incorporate this heuristic in our implementation.

GS_PQO (QueryTemplate QT, ErrorBound θ)

1. $\rho_t = \theta$ (Class I optimizers) | $\rho_t = 2 * \theta$ (Class II optimizers)
2. Optimize the points in the initial low-resolution grid.
3. Calculate ρ for each rectangle using Equation 4.
4. Organize the rectangles in a max-Heap based on their ρ values.
5. for the rectangle R_{top} at the top of the heap
6. if $\rho(R_{top}) \leq \rho_t$, Go to Step 14
7. else
8. Extract R_{top} from the heap.
9. Apply PQO inference to mid-points of qualifying edges of R_{top}. Optimize all the remaining mid-points.
10. Split R_{top} into four equal rectangles.
11. Compute ρ values for the smaller rectangles.
12. Insert the new rectangles into the heap.
13. Return to Step 5.
14. while the heap is not empty,
15. Extract R_{top} from the heap.
16. Select a plan at random from the edge end points and assign it to the mid point.
17. Recursively split the rectangles until all points inside R_{top} are processed.
18. End Algorithm GS_PQO

Figure 9: The GS_PQO Algorithm

Final Inference. As mentioned earlier, GS_PQO uses the PQO-based inference technique within each rectangle until its plan richness metric goes below the ρ_t threshold. After the threshold is crossed, there may still be unassigned points within the rectangle. These are handled as follows in the final inference phase: The same PQO-based inference scheme is used with the only difference being that whenever an edge has different end-points, then the plan assignment of the mid-point is done by randomly choosing one of the end-point plans, rather than resorting to explicit optimization.

The complete GS_PQO algorithm is shown in Figure 9.

2.2 Class II Optimizers

In the algorithms described above for the Class I optimizers, we run into situations wherein we are forced to pick from a set of equivalent candidate plans in order to make an assignment for an unoptimized query point. For example, in the RS_NN approach, if there are multiple nearest neighbors at the same distance. Similarly, in the GS_PQO approach, when the ρ value of a rectangle goes below the threshold and there remain unassigned internal points. The strategy followed is to make a random choice from the closest neighboring plans.

For Class II optimizers, however, which offer a "foreign plan costing" (FPC) feature, we can make a more informed selection: Specifically, cost all the candidate plans at the query point in question, and assign it the lowest cost plan. This method significantly helps in reducing the plan-location error, since it enables precise demarcation of the boundaries between plan optimality regions. A direct fallout obtained through our empirical investigation is that the value of ρ_t, which was set equal to θ for Class I optimizers, can be relaxed to $2 \times \theta$ while still maintaining the same accuracy characteristics, in the process noticeably lowering overheads.

Another point to be noted here is that plan-costing is much cheaper than the optimizer's standard optimal-plan-searching process [15], and hence the overheads incurred through costing are negligible compared to those incurred through optimization. In our experience, the overhead ratio of plan-costing to plan-searching is around **1:10** in the commercial optimizers, while in our implementation of this feature in PostgreSQL, it is close to **1:100**.

DiffGen (QueryTemplate QT)

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

Figure 10: The DiffGen Algorithm

2.3 Class III Optimizers

The algorithms discussed thus far minimize the number of explicit optimizations performed by assuming certain properties of the plan diagram and using these properties to infer plan assignments from the optimized query points. We now move on to presenting for the Class III optimizers, the DiffGen algorithm, which can be used to efficiently generate *completely accurate* plan diagrams. Subsequently, we provide a variant, the ApproxDiffGen algorithm, which trades error, based on the user's bound, for reduction in optimization effort. Both algorithms utilize the foreign-plan-costing (FPC) and plan-rank-list (PRL) features offered by the Class III optimizer API. Specifically, it is assumed that for each query point, the optimizer provides both the best plan and the second-best plan. As mentioned in the Introduction, this is a feature that can be easily incorporated in today's systems with only marginal changes to the codebase – the details are available in [7].

2.3.1 The DiffGen Algorithm

The DiffGen algorithm for a 2D query template is shown in Figure 10. The algorithm starts with optimizing the query point $q(x_{min}, y_{min})$ corresponding to the bottom-left query point in the plan diagram. Let p_1 be the optimizer-estimated optimal plan at q, with cost $c_1(q)$, and let p_2 be the *second best* plan, with cost $c_2(q)$. We then assign the plan p_1 to all points q' in the *first quadrant* relative to q as the origin, which obey the constraint that $c_1(q') \leq c_2(q)$. After this step is complete, we then move to the next unassigned point in row-major order relative to q, and repeat the process, which continues until no unassigned points remain.

This algorithm is predicated on the *Plan Cost Motonicity* (PCM) assumption that the cost of a plan is monotonically non-decreasing throughout the selectivity space, which is true in practice for most query templates [12]. (The handling of cases where the PCM assumption does not hold is discussed in [7].)

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

THEOREM 1. *The plan assigned by DiffGen to any point in the approximate plan diagram* A *is exactly the same as that assigned in* P.

PROOF. Let $P_o \subseteq$ P be the set of points which were optimized. Consider a point $q' \in$ P $\setminus P_o$ with a plan p_1. Let $q \in P_o$ be the point that was optimized when q' was assigned the plan p_1. Let p_2 be the second best plan at q.

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

2.3.2 *The ApproxDiffGen Algorithm*

While DiffGen always ensures zero error, we now investigate the possibility of whether it is possible to utilize the permissible error bound of θ to further reduce the computational overheads of DiffGen. To this end, we propose the following ApproxDiffGen algorithm: The plan assignment constraint $c_1(q') \leq c_2(q)$ is relaxed to be $c_1(q') \leq (1+\gamma)c_2(q)$ with $(\gamma > 0)$, resulting in fewer optimizations being required to fully assign plans in the diagram. The choice of γ is a function of θ and μ_1, the slope of the cost function c_1 at q. Our empirical assessment indicates that setting $\gamma = 0.1 * \mu_1 * \theta$ (e.g. with $\theta = 10\%$ and $\mu = 1$, $\gamma = 0.01$) is usually sufficient to both meet the error requirements and simultaneously significantly reduce the overheads. For example, $\theta = 10\%$ can be achieved with only around **1%** overheads, as seen in the following section.

3. EXPERIMENTAL RESULTS

The testbed used in our experiments is the Picasso optimizer visualization tool [23], executing on a Sun Ultra 20 workstation equipped with an Opteron Dual Core 2.5GHz processor, 4 GB of main memory and 720 GB of hard disk, running the Windows XP Pro operating system. The experiments were conducted over plan diagrams produced from a variety of two, three, and four-dimensional **TPC-H** [28] and **TPC-DS** [29] based query templates. In our discussion, we use QT*x* to refer to a query template based on Query *x* of the TPC-H benchmark, and DSQT*x* to refer to a query template based on Query *x* of the TPC-DS benchmark. The TPC-H database was of size 1GB, while the TPC-DS database occupies 100GB. The plan diagrams were generated with a variety of industrial-strength database query optimizers – we present representative results here for a commercial optimizer anonymously referred to as OptCom, and a public-domain optimizer, referred to as OptPub. Query points were uniformly distributed over the selectivity space.

In the remainder of this section, we evaluate the various approximation strategies with regard to their computational efficiency, given user-specified bounds for plan-identity and plan-location error. The bounds we consider here are $\theta = 10\%$ and $\theta = 1\%$.

Our analysis was carried out over an extensive suite of query templates. However, we selectively present results here for "challenging" plan diagrams that feature a sufficiently rich set of plans (≥ 20 plans) and involve a large computational overhead (≥ 3 hrs). The full set of results is available in [7].

3.1 Class I Optimizers

We start with evaluating the performance of the two algorithms applicable to Class I optimizers, namely, RS_NN and GS_PQO. In the RS_NN algorithm, as mentioned earlier, the parameter δ, which specifies the transition of the algorithm from Stage 1 to Stage 2, is set to 0.3, while the sample size increments are 1% of the space. For the GS_PQO algorithm, the resolution of the initial grid along each dimension is set to 10% of the resolution at which the plan diagram is to be generated. As an example, to approximate a 2D plan diagram with 300×300 resolution, we set the initial sample size of RS_NN to 900 and the initial grid of GS_PQO to 30×30.

Error Bound = 10%. For the above framework, Table 2 shows the algorithmic efficiency of the RS_NN and GS_PQO algorithms relative to the brute-force exhaustive approach for a variety of multi-dimensional query templates, under a $\theta = 10\%$ constraint. The efficiency is presented both in terms of actual time, as well as in terms of the number of optimizations that were carried out. The bracketed numbers in the *TimeTaken* columns indicate the percentage time taken relative to the exhaustive approach.

We see in Table 2 that the RS_NN algorithm requires a substantial amount of time, or equivalently, number of optimizations, to generate the approximate plan diagram. For example, with the 3D QT9 template at a resolution of 100 per dimension, RS_NN takes about 27% of the exhaustive time. On the other hand, GS_PQO exhibits a much better performance, requiring only 10% overheads – in fact, our experience has been that it needs less than 15% of the exhaustive time *across all templates*. Moreover, as can be seen from Table 2, we have also produced an approximate plan diagram for the 3D QT8 template at a resolution of 300 per dimension, corresponding to *27 million query points* in only 2 days with GS_PQO – the estimated generation time with the brute-force approach is 4 months!

We see that the estimators designed for RS_NN and GS_PQO almost always result in meeting the user's error bounds or being in their close proximity. However, for the 3D query template QT9, the ϵ_L value reaches 16% with the default ρ_t setting of 0.1. Bringing it down to 10% requires lowering the ρ_t to 0.08, with the overhead going up by 3%. In our future work, we plan to investigate automated schemes for setting the appropriate value of ρ_t.

Turning our attention to Table 3, which repeats the above experiment on the TPC-DS database, we see that the results are even more striking. RS_NN incurs large overheads in general, typically around 40%, whereas GS_PQO again does not exceed 15%.

An interesting point to note in both these tables is that the optimization percentages are virtually identical to the time percentages. This means that the inference mechanisms of NN and PQO take insignificant time as compared to making optimizer calls.

Error Bound = 1%. When the user's error constraint is tightened from 10 percent to 1 percent, the resulting algorithmic performance is shown in Table 4. Only GS_PQO is shown since for this stringent constraint, the RS_NN algorithm tends to optimize almost the entire space. Further to make the 1% error bound meaningful, we have considered only plan diagrams having around or over 100 plans. It can be seen from the table that by optimizing around 40% of the points, GS_PQO is able to generate extremely accurate approximate plan diagrams.

Cost Increase induced by Approximation. A legitimate concern in generating and using approximate plan diagrams is the following: In the erroneous locations, where a different plan has been assigned as compared to the original diagram, is it possible that the substitute plan's (estimated) cost performance is significantly worse than that of the original choice? Our experience is that the cost increase is only a few percent – this is quantified below in Table 5, which shows the maximum cost increase incurred in the erroneous locations, for a representative set of query templates. The complete set of experiments is available in [7].

3.2 Class II Optimizers

We now move on to demonstrate how the FPC feature, provided by Class II optimizers, can be used to improve the performance of GS_PQO. Tables 6 and 7 show the effort required by GS_PQO for obtaining approximate plan diagrams with $\theta = 10\%$ on the TPC-H and TPC-DS benchmarks, respectively. We see here that GS_PQO

Dimension / Resolution	Query Temp-late	No. of Plans	Exhaustive Generation Time	Approximation Time Taken		Optimizations Required (%)		RS_NN Error (%)		GS_PQO Error (%)	
				RS_NN	GS_PQO	RS_NN	GS_PQO	ϵ_I	ϵ_L	ϵ_I	ϵ_L
2D: 300X300	QT2	76	9.6 hrs	2 hrs (23%)	20 mins (4%)	23 %	4 %	11 %	8 %	4 %	11 %
	QT5	31	8.3 hrs	0.6 hrs (7%)	15 mins (3%)	7 %	3 %	10 %	3 %	4 %	10 %
	QT8	92	10.5 hrs	3.7 hrs (35%)	44 mins (7%)	35 %	7 %	10 %	1 %	3 %	10 %
	QT9	91	1 day 3 hrs	9 hrs (33%)	1.4 hrs (5%)	33 %	5 %	10 %	2 %	5 %	9 %
	QT10	31	5 hrs	0.4 hrs (8%)	9 mins (3%)	8 %	3 %	10 %	2 %	3 %	10 %
	QT20	46	1 day 7 hrs	7.5 hrs (25%)	1.3 hrs (4%)	25 %	4 %	4 %	9 %	4 %	10 %
	QT21	48	5 hrs	0.7 hrs (14%)	8 mins (3%)	14 %	3 %	4 %	4 %	4 %	10 %
2D: 1000X1000	QT8	132	6 days	29 hrs (21%)	4.2 hrs (3%)	21 %	3 %	10 %	6 %	2 %	8 %
	QT16	25	16 hrs	2 hrs (10%)	9 mins (1%)	10 %	1 %	8 %	6 %	8 %	10 %
	QT21	58	2 days 6 hrs	2.7 hrs (5%)	32 mins (1%)	5 %	1 %	12 %	9 %	2 %	6 %
3D: 100X100X100	QT8	190	6 days 10 hrs	1.6 days (26%)	16 hrs (10%)	26 %	10 %	11 %	2 %	8 %	10 %
	QT9	404	10 days	64 hrs (27%)	24 hrs (10%)	27 %	10 %	10 %	6 %	8 %	16 %
	QT21	130	3 days	15 hrs (21%)	5.8 hrs (8%)	21 %	8 %	2 %	4 %	10 %	13 %
3D: 300X300X300	QT8	314	4 months (est)	–	2 days (2%)	–	2 %	–	–	–	–
4D: 30X30X30X30	QT8	243	5 days	23 hrs (19%)	15 hrs (12%)	19 %	12 %	12 %	9 %	4 %	9 %

Table 2: Approximation Efficiency for Class I optimizers with TPC-H database ($\theta = 10\%$) [OptCom]

Dimension / Resolution	Query Temp-late	No. of Plans	Exhaustive Generation Time	Approximation Time Taken		Optimizations Required (%)		RS_NN Error (%)		GS_PQO Error (%)	
				RS_NN	GS_PQO	RS_NN	GS_PQO	ϵ_I	ϵ_L	ϵ_I	ϵ_L
2D: 100X100	DSQT 17	39	6.7 hrs	2.6 hrs (39%)	40 mins (10%)	39 %	10 %	8 %	8 %	5 %	5 %
	DSQT 25	33	7 hrs	4.6 hrs (65%)	46 mins (11%)	65 %	11 %	10 %	9 %	4 %	4 %
	DSQT 25a	51	6.5 hrs	1.5 hrs (24%)	42 mins (11%)	24 %	11 %	12 %	11 %	12 %	3 %
	DSQT 25b	45	7.3 hrs	2.6 hrs (36%)	48 mins (11%)	36 %	11 %	9 %	9 %	5 %	4 %
2D: 300X300	DSQT 18	81	22.5 hrs	8.7 hrs (38%)	3 hrs (13%)	38 %	13 %	12 %	8 %	2 %	3 %
	DSQT 19	42	16.2 hrs	1 hr (7%)	58 mins (6%)	7 %	6 %	7 %	7 %	5 %	6 %

Table 3: Approximation Efficiency for Class I optimizers with TPC-DS database ($\theta = 10\%$) [OptCom]

Dimension/ Resolution	Query Template	No. of Plans	Exhaustive Generation time	Time taken by GS_PQO	Optimizations performed by GS_PQO (%)	GS_PQO Error (%)	
						ϵ_I	ϵ_L
2D: 300X300	QT 8	92	10.5 hrs	3.6 hrs (35%)	35 %	0 %	0.25 %
	QT 9	91	1 day 3 hrs	7.3 hrs (30%)	30 %	0 %	2 %
2D: 1000X1000	QT 8	132	6 days	1 day 18 hrs (30%)	30 %	2 %	1 %
3D: 100X100X100	QT 8	190	6 days 10 hrs	1 day 14 hrs (25%)	25 %	0.5 %	1.5 %
	QT 9	404	10 days	4 days (40%)	40 %	1 %	1 %
	QT 21	130	3 days	11 hrs (16%)	16 %	0.77 %	1.5 %

Table 4: Approximation Efficiency for Class I optimizers with TPC-H database ($\theta = 1\%$) [OptCom]

often reduces the approximation overheads by a significant fraction as compared to the corresponding numbers in Tables 2 and 3, testifying to the utility of FPC. As a case in point, the 10% overhead incurred by the 3D:100x100x100 flavor of QT8 with the Class I optimizer is reduced to 4% with the Class II optimizer.

With an error bound of 1%, however, the role of FPC becomes limited since inference is rare, and therefore the diagram approximation time is similar to that seen for Class I optimizers (Table 4).

Dimension/ Resolution	Query Template	Maximum % error
2D: 300X300	QT8	8
2D: 300X300	QT9	10
3D: 100X100X100	QT8	14

Table 5: Maximum Cost Increase due to Approximation

3.3 Class III Optimizers

Turning our attention to Class III optimizers, we now evaluate the two algorithms, DiffGen and ApproxDiffGen for TPC-H benchmark queries. Due to space limitations, the complete set of results, including those on the TPC-DS benchmark, is deferred to [7]. For this experiment, the OptPub engine was modified to (a) implement the FPC feature internally, and (b) to return the second best plan along with the optimal plan when the "explain"command is executed.

DiffGen. The performance results for DiffGen are shown in Table 8 – due to the change in database engine from OptCom to OptPub, the set of query templates with "challenging" plan diagrams differs as compared to our earlier results. We observe that DiffGen usually requires at most 10% optimizations to generate a *completely accurate* plan diagram for all query templates, except those based on Query 8, the reason for which is discussed below. The good performance of DiffGen can be attributed to the following: Along with the optimizations being performed at select points, all points are costed exactly once. Further, since the FPC feature is internalized in the optimizer, the ratio of plan-costing to plan-searching is approximately 1:100, making the overheads incurred very small.

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

ApproxDiffGen. Turning our attention to the ApproxDiffGen algorithm, whose performance is presented in Table 9 for a 10% error

Dimension/ Resolution	Query Template	No. of Plans	Exhaustive Generation time	Time taken by GS_PQO	Optimizations performed by GS_PQO (%)	GS_PQO Error (%)	
						ϵ_I	ϵ_L
2D: 300X300	QT2	76	9.6 hrs	11 mins (2%)	2 %	8 %	8 %
	QT5	31	8.3 hrs	10 mins (2%)	2 %	3 %	4 %
	QT8	92	10.5 hrs	12 mins (2%)	2 %	1 %	4 %
	QT9	91	1 day 3 hrs	32 mins (2%)	2 %	2 %	4 %
	QT10	31	5 hrs	5 mins (2%)	2 %	10 %	5 %
	QT20	46	1 day 7 hrs	28 mins (2%)	2 %	6 %	10 %
	QT21	48	5 hrs	4 mins (1%)	1 %	6 %	8 %
2D: 1000X1000	QT8	132	6 days	3.8 hrs (3%)	3 %	6 %	4 %
	QT16	25	16 hrs	9 mins (1%)	1 %	8 %	5 %
	QT21	58	2 days 6 hrs	32 mins (1%)	1 %	9 %	10 %
3D: 100X100X100	QT8	190	6 days 10 hrs	6.1 hrs (4%)	4 %	11 %	8 %
	QT9	404	10 days	21.6 hrs (9%)	9 %	6 %	4 %
	QT21	130	3 days	3.5 hrs (5%)	5 %	4 %	4 %
3D: 300X300X300	QT8	314	4 months (est)	2 days (2%)	2%	–	–
4D: 30X30X30X30	QT8	243	5 days	15 hrs (12%)	12 %	4 %	4 %

Table 6: Approximation Efficiency for Class II optimizers with TPC-H database ($\theta = 10\%$) [OptCom]

Dimension/ Resolution	Query Template	No. of Plans	Exhaustive Generation time	Time taken by GS_PQO	Optimizations performed by GS_PQO (%)	GS_PQO Error (%)	
						ϵ_I	ϵ_L
2D:100X100	DSQT 17	39	6.7 hrs	20 mins (5%)	5 %	10 %	10 %
	DSQT 25	33	7 hrs	45 mins (10%)	10 %	6 %	8 %
	DSQT 25a	51	6.5 hrs	42 mins (10%)	10 %	10 %	12 %
	DSQT 25b	45	7.3 hrs	30 mins (7%)	7 %	10 %	6 %
2D:300X300	DSQT 18	81	22.5 hrs	1.2 hrs (5%)	5 %	9 %	8 %
	DSQT 19	42	16.2 hrs	24 mins (3%)	3 %	7 %	8 %

Table 7: Approximation Efficiency for Class II optimizers with TPC-DS database ($\theta = 10\%$) [OptCom]

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

Table 8: Zero-error Efficiency for Class III Optimizers with TPC-H database [OptPub]

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

Table 9: Approximation Efficiency for Class III Optimizers with TPC-H database ($\theta = 10\%$) [OptPub]

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

In Table 9, the maximum number of plans produced by a query template is only 49 which is much below 100 – therefore, the performance of ApproxDiffGen for $\theta = 1\%$ is equivalent to that of DiffGen, which can be viewed as ApproxDiffGen with $\theta = 0\%$.

A related point to note is that unlike the Optimizer I and II classes where the time and optimization overheads are virtually identical, here the time overheads are a little more than that of optimization. The reason is that although FPC is very cheap, since it has to be invoked for a very large number of points, a small but perceptible time overhead results.

4. CONCLUSIONS

We have investigated in this paper the efficient generation of approximate plan diagrams, a key resource in the analysis and redesign of modern database query optimizers. Based on the optimizer's API capabilities, we made a partitioning into three different classes of optimizers, and developed appropriate approximation techniques for each class. For Class I, which only provides the optimal plan, our experimental results showed that the GS_PQO algorithm, which combines grid sampling with PQO inference at the micro level, performed very adequately requiring less than 15% overheads as compared to the exhaustive approach, for an error

bound of 10%. These overheads came down to 10% when the same algorithm was used in Class II optimizers, due to their additional FPC feature. Finally, for Class III systems, we proved that the DiffGen algorithm produced zero errors and was generally able to do so incurring overheads of less than 10%. However, it performs poorly for query templates that have the second-best plan being very close to the optimal choice over an extended region. Finally, the ApproxDiffGen algorithm traded error for performance, and was able to satisfy the 10% error bound with less than 5% optimizations. It was also able to adequately handle the problem query templates of DiffGen.

In summary, our work has shown that it is indeed possible to efficiently generate close approximations to high-dimension and high-resolution plan diagrams, with typical overheads being an *order of magnitude lower* than the brute-force approach. We hope that our results will encourage all database vendors to incorporate the foreign-plan-costing and/or plan-rank-list features in their optimizer APIs.

All the plan diagrams featured in this paper were produced using a uniform distribution of the locations of query points. We are currently investigating the extension of our approximation techniques to plan diagrams generated with exponential distributions of query points. Further, our algorithms feature tuning parameters that have been set after considerable empirical testing. These settings may be a function of the specific optimizer engines and database environments assessed in our experiments – in our future work, we plan to investigate the portability of these settings over a broader spectrum of engines and environments.

Acknowledgements. This work was supported in part by research grants from IBM, Microsoft and Google.

5. REFERENCES

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

[2] M. Charikar, S. Chaudhuri, R. Motwani and V. Narasayya,"Towards Estimation Error Guarantees for Distinct Values", *Proc. of ACM Symp. on Principles of Database Systems (PODS)*, 2000.

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

[7] A. Dey, S. Bhaumik, Harish D. and J. Haritsa "Efficient Generation of Approximate Plan Diagrams", *Tech. Rep. TR-2008-01, DSL/SERC, Indian Inst. of Science*, 2008. `http://dsl.serc.iisc.ernet.in/publications/report/TR/TR-2008-01.pdf`

[8] 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.

[9] R. Gonzalez and R. Woods, *Digital Image Processing*, Pearson Prentice Hall, 2007.

[10] P. Haas and L. Stokes. "Estimating the number of classes in a finite population". *In Journal of the American Statistical Association, 93*,1998.

[11] P. Haas, J. Naughton, S. Seshadri and L. Stokes,"Sampling-Based Estimation of the Number of Distinct Values of an Attribute", *Proc. of 21st Intl. Conf. on Very Large Databases (VLDB)*, 1995.

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

[13] Harish D., P. Darera and J. Haritsa, "Identifying Robust Plans through Plan Diagram Reduction", *Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2008.

[14] 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.

[15] 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.

[16] 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.

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

[18] V. Prasad, "Parametric Query Optimization: A Geometric Approach", *Master's Thesis, Dept. of Computer Science & Engineering, IIT Kanpur*, April 1999.

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

[20] 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.

[21] N. Reddy, "Next-Generation Relational Query Optimizers", *Master's Thesis, Dept. of CSA, Indian Institute of Science*, June 2005, `http://dsl.serc.iisc.ernet.in/publications/thesis/naveen.pdf`.

[22] P. Tan, M. Steinbach and V. Kumar, *Introduction to Data Mining*, Addison-Wesley, 2005.

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

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

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

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

[27] `http://postgresql.org`

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

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