Producing on the fly anorexic plan diagrams using AniPQO

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

Faculty of Engineering

BY

Mahajan Chetas Subhash

Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012 (INDIA)

JULY 2015

Declaration of Originality

I, **Mahajan Chetas Subhash**, with SR No. **04-04-00-10-41-13-1-10312** hereby declare that the material presented in the thesis titled

Producing on the fly anorexic plan diagrams using AniPQO

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2013-15**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

DEDICATED TO

My Family

for continuous support and encouragement

Acknowledgements

I am deeply grateful to Prof. Jayant Haritsa for his unmatched guidance, enthusiasm and supervision. He has always been a source of inspiration for me. I have been extremely lucky to work with him.

Also, I am thankful to Anshuman Dutt for all those long discussions and suggestions. It had been a great experience to work with him.

My sincere thanks goes to my fellow lab mates for all the help and suggestions. Also I thank my CSA friends who made my stay at IISc pleasant, and for all the fun we had together.

Finally, I am indebted with gratitude to my parents and younger brother for their love and inspiration that no amount of thanks can suffice. This project would not have been possible without their constant support and motivation.

Abstract

A "plan diagram" is a pictorial enumeration of the execution plan choices of a database query optimizer over the relational selectivity space. These diagrams are often remarkably complex and dense, with a large number of plans covering the selectivity space, showing that current optimizer make extremely fine-grained plan choices. However, they can often be reduced to much simpler pictures termed as "anorexic plan diagrams" with significantly fewer plans, with bounded degradation in the query processing quality. This can be achieved by applying an anorexic reduction technique to the plan diagram but the associated processing overheads may hinder the practical relevance. There is a heuristic approach, known as AniPQO, that avoids the brute force exploration of the selectivity space, and identifies a subset of plans (from plan diagram) using geometric decomposition of selectivity space. Thus, it allows us to produce an approximation to the plan diagram but does not ensure bounded degradation in query processing quality. When evaluated on the top of small home-grown Volcano based optimizer, AniPQO was found to identify anorexic set of plans, with significantly lesser overheads and without significant (not bounded) degradation in the query processing quality.

This work aims to evaluate AniPQO on modern industrial-strength optimizer and to check whether it can be used as "on the fly" technique to produce anorexic plan diagrams. We have evaluated AniPQO using TPC-H and TPC-DS benchmarks on SQL Server 2008, which provides plan forcing feature. Empirical results show that AniPQO also gives anorexic set of plans on such industrial-strength optimizer and it can be used to directly produce anorexic plan diagrams. We have also visualized anorexic plan diagrams associated with AniPQO, using tool Picasso, as part of this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In modern database systems, data processing is done by executing SQL queries. These SQL queries are declarative and *query optimizer* module is employed to consider different strategies, called "plans", to execute. It evaluates each plan by estimating its expected cost in terms of query response time. Estimated cost of plan is generally function of *Selectivity* which is the estimated number of rows of each relation relevant to producing the final result. In [7], a fresh perspective is created on the behaviour of modern query optimizers by developing the notion of a "plan diagram". A plan diagram is a pictorial enumeration of the execution plan choices made by a database query optimizer over the relational selectivity space. In a nutshell, plan diagrams pictorially capture the geometries of the optimality regions of the "parametric optimal set of plans" (POSP), which covers the whole selectivity space. Plan diagrams can be generated from the visualization platform "Picasso" [7].

Plan diagrams are often remarkably complex and dense, with a large number of plans covering the space, showing that current optimizers make extremely fine-grained plan choices. However, if users were willing to tolerate a minor cost increase of at most $\lambda\%$ for any query point in the diagram, plan diagram can be reduced to much simpler picture, termed as "anorexic plan diagram" or "reduced plan diagram", with significantly fewer plans, without materially affecting the query processing quality [2]. The set of these fewer plans now covers the whole selectivity space and the set of plans is termed as "Reduced-POSP", with tolerance of λ. These reduced plan diagrams can be produced by using *anorexic reduction* technique, CostGreedy (**CG**), described in [2] and its variant **CG-FPC** described in [4]. But such reduction techniques are *post-processing* techniques, which need *plan diagrams* beforehand and even plan diagram is generated by making many "optimization calls" to the query optimizer. CG is very fast, where

CG-FPC is extremely slow. But CG-FPC gives better reduction than CG.

Previously, a heuristic technique called as "Almost Non-Intrusive Parametric Query Optimization" (AniPQO) was proposed in [6]. It takes user-defined threshold, λ, as a input and outputs a subset of POSP, $AniPOSP$, where for every query point in the selectivity space, there exists at least one plan in $AniPOSP$ which is either optimal or co-optimal, with cost-increment of atmost λ% threshold. AniPQO gives the guarantee, that the $AniPOSP$ is actually a *Reduced-POSP* with the user-defined threshold λ, in case of "Linear Cost Functions" (LCF) where plan cost functions are linear with respect to selectivities. It doesn't give this guarantees in case of "Non-Linear Cost Functions" (NLCF). But empirical results show that cost-degradation is not significantly large also in case of NLCF.

AniPQO utilizes "foreign plan costing" (**FPC**), in which query along with certain execution plan is given to optimizer and optimizer returns the cost of a given plan. AniPQO was evaluated on top of small home-grown Volcano based optimizer known as Pyro developed at IIT, Bombay. This optimizer follows non-linear cost model [3] and supports FPC feature [6]. AniPQO does more number of FPC calls than optimization calls, where optimization is the optimal-plan-selection-process from the exponential search-space and FPC is the process of just estimating the cost of a given plan. Hence, FPC calls are very cheap as compared to optimization calls [6], which makes AniPQO to perform *extremely fast* as compared to brute-force approach.

1.2 Motivation

Pyro [9] is not an *industrial-strength* optimizer. At the time when this technique was proposed, no commercial industrial-strength optimizer were supporting FPC feature and so, this technique was not evaluated on such optimizers. Today, many such modern commercial industrial-strength database engines support FPC feature. This work is to evaluate AniPQO on commercial industrial-strength optimizer to check how well AniPQO performs. If AniPQO performs well on commercial industrial-strength optimizer, that is, $AniPOSP$ can be considered as a Reduced-POSP, unlike post-processing techniques which need plan diagrams beforehand, reduced plan diagrams can be generated more efficiently and *on the fly* without any need of plan diagrams.

1.3 Contributions

Current implementation of AniPQO[1] is tightly coupled with Pyro. We divide our contributions in two parts as follows:

[1]We are very thankful to Arvind Hulgeri for providing us AniPQO's implementation

1.3.1 Implementation

We have developed following as part of this work:

- **Interface with SQL Server:** We have decoupled AniPQO's module from Pyro and developed an interface to connect with SQL Server by using ODBC.
- **Flexible input:** Current implementation of AniPQO works on fixed integer-parameter-space generated by using metadata of database. We have modified its implementation to work on grid of selectivity space of any resolution. Now, this grid has selectivity constants which can be generated by user. We have used selectivity constants generated directly from Picasso [7].
- **Integration with Picasso:** AniPQO returns *AniPOSP*, which is used at runtime to find the best plan at a given query point. We have done the visualization of the reduced plan diagram by selecting best plans only from *AniPOSP* at each query location. This visualization is done using Picasso. We have also enhanced Picasso, latest version 2.1, by integrating AniPQO with it. Reduced plan diagrams can now be *directly* generated using Picasso with the help of AniPQO.

1.3.2 Evaluation

We have done following for evaluation of AniPQO:

- **Error metrics:** In [6], error metric considered for evaluation of AniPQO was "Maximum degradation of cost". But this metric doesn't measure portion of selectivity space which is actually harmed, that is, number of query points at which *cost-degradation percentage* is greater than λ. We have introduced additional error metric, termed as "Harmed location percentage", refers to the percentage of selectivity space which is harmed. We considered maximum and average degradation of cost only at harmed locations and termed them as "Max harm" and "Average harm" respectively.
- **Evaluation on TPC-H and TPC-DS:** We have evaluated AniPQO on TPC-H benchmark database using benchmark query templates with two different physical designs of our database engine, that is, **PrimaryKey(PK)** and **AllIndex(AI)**. We have also evaluated AniPQO on TPC-DS benchmark database with physical design PK.
- **Comparison of AniPQO with CG-FPC:** We have done comparison of AniPQO with CG-FPC using their reduced plan diagrams. We have a strong claim with a lot of evidence, that *AniPQO is as fast as CG and as reductive as CG-FPC.*

- **Scalability on higher dimensions:** We have also checked scalability of AniPQO in terms of number of optimization calls increased with the increase of dimension.

1.4 Organization

The remainder of this thesis is organized as follows: Chapter 2 provides the background which includes brief outline of plan diagram, anorexic reduction and AniPQO. Problem identification and definition are given in Chapter 3. In Chapter 4, implementation-side contributions of our work are discussed. The various evaluation scenarios of AniPQO are discussed in Chapter 5. In Chapter 6, conclusion of this work and the outline of future work are provided.

Chapter 2

Background

In this chapter, we present necessary details of plan diagrams, reduction techniques and AniPQO.

2.1 PQO, POSP and Plan diagrams

The cost of a plan depends on various database and system parameters. The database parameters include selectivity of predicates and sizes of the relations. The system parameters include available memory, disk bandwidth and latency. Exact values of these parameters may not be known at compile time. Optimizing a query afresh each time it is executed can add substantially to the cost of evaluation. To overcome this problem, "parametric query optimization" (PQO) optimizes a query into a number of candidate plans, each optimal for some region of the parameter space [6]. This set of optimal plans is termed as "parametric optimal set of plans" (POSP) which covers whole selectivity space. This POSP can be found by querying optimizer for every combination of parameters, which is a *brute-force* way, by using the notion of "plan diagram". Specifically, a plan diagram is a *visual* representation of the plan choices made by the optimizer over an input parameter space, whose dimensions may include database, query and system-related features. In a nutshell, plan diagrams pictorially capture the geometries of the optimality regions of the POSP [5].

To make these notions clear, consider parametric query template, TPC-H QT8 [10], shown in Figure 2.1, which is based on TPC-H Query 8. The template defines a relational *selectivity space* on the SUPPLIER and LINEITEM relations, with the selectivity variations specified through the s_acctbal :varies and l_extendedprice :varies predicates, respectively.

The associated plan diagram for QT8[1], with 300 × 300 resolution grid granularity and uniform distribution, on SQL Server 2008 is shown in Figure 2.2. Here, grid granularity of

[1]The figure in this report should ideally be viewed from color copy to clearly register the feature

```
select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(volume)
from (select YEAR(o_orderdate) as o_year, l_extendedprice * (1 - l_discount) as volume,
n2.n_name as nation
        from part, supplier, lineitem, orders, customer, nation n1, nation n2, region
        where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey
        and o_custkey = c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey
        = r_regionkey and r_name = 'AMERICA' and s_nationkey = n2.n_nationkey and
        o_orderdate between '1995-01-01' and '1996-12-31' and p_type = 'ECONOMY AN-
        ODIZED STEEL'
        and s_acctbal :varies and l_extendedprice :varies
        ) as all_nations
group by o_year
order by o_year
```

Figure 2.1: Example Query Template: QT8

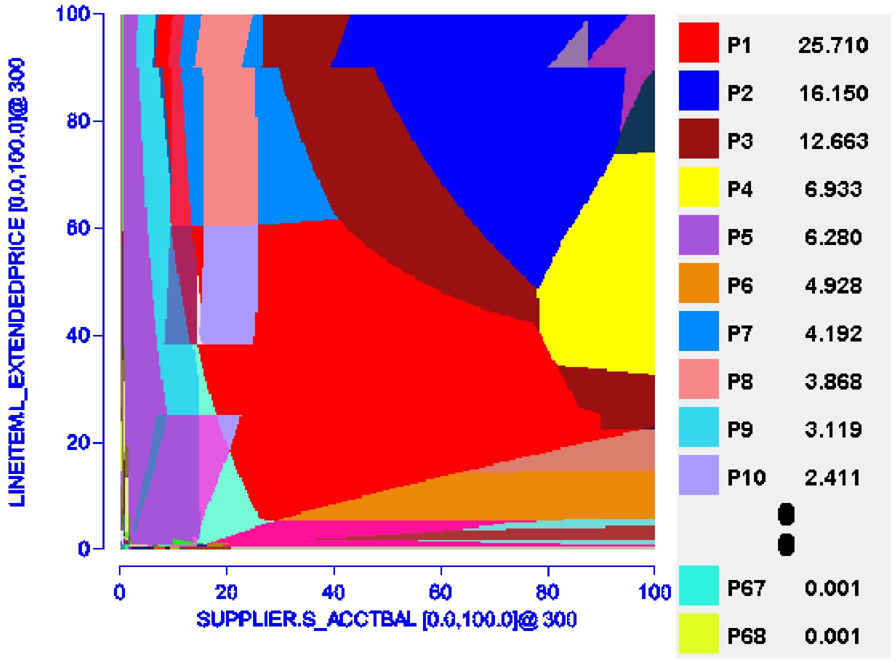


Figure 2.2: Plan diagram of QT8 (# plans 68)

resolution 300 × 300 means, 300 instances on each of 2 dimension of this query template are distributed across the selectivity space. Two dimensions are varying predicates in QT8, those are, SUPPLIER.S_ACCTBAL and LINEITEM.L_EXTENDEDPRICE. In this picture, each colored region represents a specific plan and a set of 68 different optimal plans, P1 through P68, 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 25.710% of the space, whereas the smallest, P68, is chosen in only 0.001% of the space.

2.2 Reduced plan diagrams and Reduced-POSP

These diagrams are often remarkably complex and dense, with a large number of plans covering the space showing that modern optimizers make extremely fine-grained plan choices. However these diagrams can be reduced to much simpler pictures by using *anorexic reduction* technique described in [2]. These simpler pictures are termed as "Anorexic plan diagrams" with significantly fewer plans, without materially affecting the query processing quality.

For example, if user were willing to tolerate a minor cost increase of atmost 20% for any query point in the diagram relative to its original (optimizer-estimated) cost, Figure 2.2 could be reduced to that shown in Figure 2.3, where only 4 plans remain - that is, most of the original plans have been "completely swallowed" by their siblings, leading to a highly reduced plan cardinality.

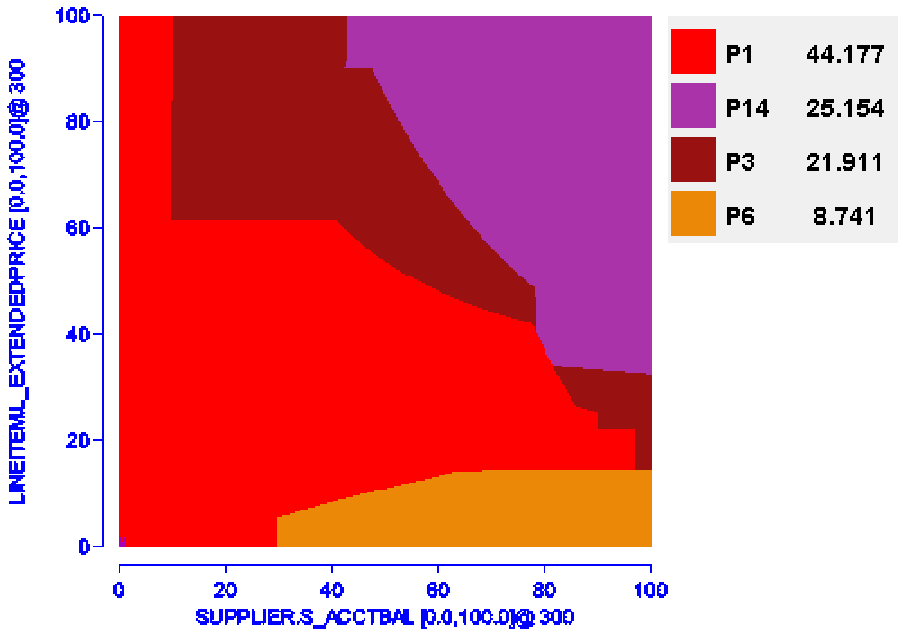


Figure 2.3: Reduced Plan diagram of QT8 using CG ($\lambda = 20\%$, # plans 4)

In this technique, POSP plans are allowed to "swallow" their sibling plans, that is, occupy their regions in the selectivity space, if the sub-optimality introduced due to these swallowings can be bounded to a user-defined threshold, λ. In [2], it was shown that even for complex OLAP queries, a λ value of 20% was typically sufficient to bring the number of POSP plans down to "anorexic levels", that is, a small absolute number within or around 10. We term this set of few plans, after reducing POSP, as "Reduced-POSP" with user-defined threshold λ, if introduced sub-optimality due to swallowings is bounded by λ.

The technique described in [2], "CostGreedy" (**CG**), is based on a conservative upper-bounding of plan costs. After generating a plan diagram, CG works on it extremely efficiently to produce reduced plan diagram and so, Reduced-POSP. There is another variant of CG, termed as **CG-FPC** mentioned in [4], which utilizes "foreign plan costing" (FPC) feature, wherein plans can be costed *outside* of their native optimality regions. It costs every plan, in plan diagram, on every query point and then does reduction. Hence, CG-FPC gives better reduction than CG, but it has enormous computational overheads as compared to CG.

2.3 AniPQO and generating Reduced-POSP directly

AniPQO solves PQO problem for general case when the cost functions may be non-linear in the given parameters. AniPQO has following interesting properties:

- AniPQO works with arbitrary non-linear and discontinuous cost functions.
- AniPQO works with arbitrary number of parameters.
- AniPQO is minimally-intrusive in the sense, that it does not need to modify the conventional query optimizer, and can merely use it as a subroutine (invoking it with different parameter values).
- AniPQO takes parameter space and user-defined threshold, λ, as a input and returns $AniPOSP \subseteq POSP$ as a output. This subset, $AniPOSP$, is indeed the Reduced-POSP with λ, that is, user-defined threshold.
- AniPQO gives guarantee to get Reduced-POSP and exact decomposition of parameter space for linear cost functions (LCF). It does not give guarantee for non-linear cost functions (NLCF). But it is empirically shown in [6], that it still gives good subset of POSP in case of NLCF, where maximum cost degradation is insignificantly larger than λ and $AniPOSP$ can still be considered as Reduced-POSP with given λ.

We have provided brief outline of AniPQO informally. One may refer the formal description of AniPQO in [6].

The Algorithm 1 is an abstraction for the procedure for finding the vertices of the parameter space decomposition induced by a plan set. It starts with empty "current set of plans" ($CSOP$) and parameter space decomposition is the parameter space polytope itself. At each step, non-optimized vertex of the decomposition is optimized. If the plan returned by optimizer is not in

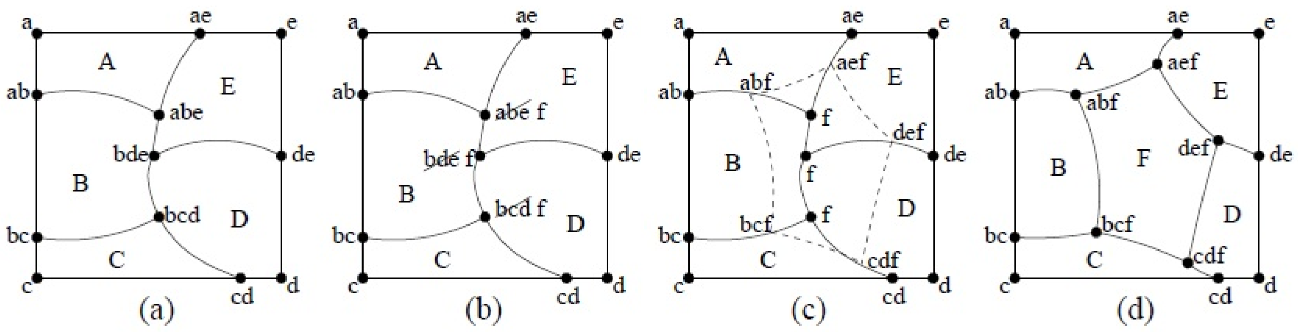


Figure 2.4: Carving out region for a new plan in $CSOP$

$CSOP$, it is added in $CSOP$ and the parameter space polytope is decomposed afresh on the new $CSOP$. This procedure is repeated till all vertices of the decomposition are optimized.

Algorithm 1 AniPQO

```
    Input: $n$ (the number of parameters), parameter space polytope /* A polytope in $\mathcal{R}^n$ */
    Output: $AniPOSP \subseteq POSP$

 1: $DecompositionVertices$ = vertices of parameter space polytope
 2: $CSOP = \emptyset$ /* Current set of optimal plans */
 3: $VerticesOptimized = \emptyset$
 4: while $DecompositionVertices - VerticesOptimized \neq \emptyset$ do
 5:     $v$ = a vertex from $DecompositionVertices - VerticesOptimized$
 6:     $p = ConventionalOptimizer(v)$ /* $p$ is one the optimal plans at $v$ */
 7:     $VerticesOptimized = VerticesOptimized \cup \{p\}$
 8:     if $p \notin CSOP$ then
 9:         $DecompositionVertices$ = vertices of parameter space decomposition induced by
    $CSOP$ /* Done by using algorithm 2 */
10:         $CSOP = CSOP \cup \{p\}$
11:     end if
12: end while
    return $CSOP$
```

Optimality threshold (λ): If the cost of a plan at a point is within a small percent (λ) of the optimal plan at the point then it is treated to be optimal plan. This threshold is taken from user. Consider an intermediate parameter space decomposition. After optimizing vertex v, a new plan $p \notin CSOP$ is found. If the cost of some plan $p' \in CSOP$ is within threshold (λ) of the cost of p at v, then v is marked as optimized and $CSOP$ along with the intermediate parameter space decomposition continue to be the same. In the case of LCF, the above procedure guarantees that at any point in the parameter space polytope, the best plan in the approximate POSP is within λ% of the optimal plan. But this bound can not be guaranteed for the case of NLCF, because of *traversal intersection* assumption between cost functions, and it can only be used as a heuristic [6].

Algorithm 2 UpdateDecompositionVertices

```
    Input: $CSOP$, $\mathcal{V}$ (the current set of decomposition vertices), $p$ (a new plan)
    Output: $\mathcal{V}$ (the updated set of decomposition vertices)

    /* Update the set of decomposition vertices $\mathcal{V}$ when a new plan $p$ is added to $CSOP$ */
 1: for each edge $(u, v)$ in edge skeleton s.t.
        $p$ is optimal (w.r.t. $CSOP \cup \{p\}$) at $v$ and
        $p$ is not optimal (w.r.t. $CSOP \cup \{p\}$) at $u$ do
 2:     $P = \mathcal{O}^{CSOP}_{\{u,v\}} \cup \{p\}$
        /* $\mathcal{O}^{CSOP}_{\{u,v\}}$ is set of plans in $CSOP$ that are optimal (within $CSOP$) at both $u$ and $v$ */
 3:     $R$ = hyper-rectangle with $u$ and $v$ as diagonal vertices
 4:     $w = FindEquiCostPoint(R, P)$
        /* Find vertex $w$ in $R$, s.t. at $w$, plans in $P$ are equi-cost; this may fail */
 5:     if previous step fails then
 6:         $R$ = parameter space hyper-rectangle
 7:         $w = FindEquiCostPoint(R, P)$
        /* This step too may fail */
 8:     end if
 9:     If vertex $w$ is found, insert $w$ in $\mathcal{V}$
10: end for
    Remove from $\mathcal{V}$ the vertices at which no plan other than $p$ is optimal from $CSOP$
```

Representation and manipulation of the decomposition: Figure 2.4 illustrates the operation of carving out the optimality region for a new plan added to $CSOP$. The regions are non-linear; defining and manipulating them exactly even for 2 parameter case is not easy. That's why those are approximated with convex polytopes which can be represented and manipulated efficiently.

An "edge skeleton" is representation of parameter space polytope by its edges and vertices. Figure 2.4 is the example of edge skeleton. Procedure in Algorithm 1 needs only set of decomposition vertices and not the complete decomposition. For each vertex in decomposition, the set of plans from $CSOP$ optimal (within $CSOP$) at the vertex is stored. This information is enough to find the edge skeleton and update the set of decomposition vertices when a new plan is added to $CSOP$. Procedure of updating set of decomposition vertices is explained in Algorithm 2

The edges defining decomposition are of two types: those on the boundary of the rectangle and those inside it. In each edge in former set, atleast one plan should be common between its two vertices (e.g. ab-bc). In each edge in later set, atleast two plans should be common between its two vertices (e.g. abe-bde). This is more formally defined in [6].

Algorithm 3 FindEquiCostPoint

```
   Input: $R$ (a hyper-rectangle), $P$ (a set of plans)
   Output: point $c \in R$ at which plans in $P$ are equi-cost

   /* Find a point in hyper-rectangle $R$ at which plans in $P$ are equi-cost */
1: Let $\mathcal{V}_R$ be the set of vertices of $R$
2: Let $\mathcal{H}_R$ be the dimension of $R$
3: Label each vertex $v \in \mathcal{V}_R$ by $\mathcal{O}_v^P$
   /* $\mathcal{O}_v^P$ is the set of plans from $P$ that are optimal (within $P$) at $v$ */
4: Let $\mathcal{U}_R^P = \cup_{v \in \mathcal{V}_R} \mathcal{O}_v^P$
   /* Each plan in $\mathcal{U}_R^P$ is optimal (within $P$) at least at one vertex of $R$ */
5: if $\mathcal{U}_R^P \neq P$ then return NULL
       /* the desired point is not found in $R$ */
6: end if
7: Let $c$ be the centre of $R$
8: if {all the plans in $P$ are equi-cost (within a threshold) at $c$ } $OR$ {$R$ can not be partitioned
   further} then return $c$
       /* the desired point is found in $R$ */
9: end if
   Partition $R$ into smaller $2^{\mathcal{H}_R}$ rectangles and apply the same procedure till we find the desired
   point
```

Updating the decomposition vertex set: When new plan is detected (not in $CSOP$), Algorithm 2 finds all *"conflicting"* edges in decomposition s.t. only new plan p is optimal (within $CSOP \cup \{p\}$) at one end v and not optimal at another end u. For example in Figure 2.4, abe-ab is conflicting edge. At each step, it creates hyper-rectangle s.t. conflicting edge as its diagonal and finds equi-cost point in this hyper-rectangle by using Algorthm 3, at which all common optimal plans at vertices of conflicting edge with p are equi-cost within threshold λ. Algorithm may or may not find equi-cost point in non-linear case. If found, then it is added to decomposition vertex set. Otherwise, it is found in whole parameter space polytope. This too may fail. Hence, AniPQO may miss some vertices in decomposition and it results in $AniPOSP \subset POSP$. But experimental results in [6] show that it still finds good subset of $POSP$. Figure 2.4 is example of one iteration in AniPQO.

Procedure of finding equi-cost point is given in Algorithm 3. Algorithm uses heuristic test, explained more formally in [6], that whether each plan in P is optimal atleast at one vertex of hyper-rectangle R. Algorithm starts with hyper-rectangle for which test evaluates positive; it partitions it and picks a partition for which test evaluates positive. If test does not evaluate positive for any partition, then it skips this hyper-rectangle. This is done recursively till the plans are approximately equi-cost at the centre of the hyper-rectangle and takes the centre as an approximation of the actual equi-cost point.

Chapter 3

Problem Framework

In this chapter, we will summarize capabilities of techniques described in previous chapter to identify problem and then we will state the problem formally.

3.1 Problem identification

Following table summarizes capabilities of CG, CG-FPC and AniPQO.

Metric	**CG**	**CG-FPC**	**AniPQO**
Guarantee of bounded cost-degradation on non-linear plan-cost-functions	Yes	Yes	No
Requires plan diagram beforehand	Yes	Yes	No
API features required	Opt	Opt + FPC	OPT + FPC
Reduction quality on industrial benchmark query templates	Anorexic	Better than CG	???
Computational overheads on industrial benchmark query templates	R^D	$R^D + P * R^D$	???
Approach	Brute-Force	Brute-Force	Not Brute-Force

Table 3.1: Summarizing capabilities of CG, CG-FPC and AniPQO

Table 3.1 shows that CG and CG-FPC give guarantee of bounded cost-degradation on non-linear plan-cost-functions, behaviour observed on modern commercial industrial-strength optimizers, where AniPQO doesn't give such guarantee. CG and CG-FPC require plan diagrams beforehand for reduction, where AniPQO doesn't require it. CG does not need FPC feature for reduction, but still needs optimization when plan diagram is produced; where CG-FPC, along with optimization, needs FPC feature for reduction. AniPQO also requires both optimization and FPC feature, but difference between CG-FPC and AniPQO is that CG-FPC uses optimization and FPC in different phases, that is, optimization while producing plan diagram and FPC while reduction; where AniPQO systematically uses optimization and FPC for exploring selectivity space. It is shown that CG gives anorexic reduction on industrial benchmark (TPC-H and TPC-DS) query templates on industrial-strength optimizers and CG-FPC gives even better reduction than CG. AniPQO was not evaluated on industrial-strength optimizer and not even "benchmark query templates". So, reduction quality given by AniPQO is unknown. Computational overheads of CG on industrial benchmark query templates are R^D, where R be resolution and D be dimension, as it needs to make R^D optimization calls for producing plan diagram before reduction; where CG-FPC makes $P * R^D$ FPC calls in reduction phase, P is the cardinality of POSP, along with R^D optimization calls like CG. It is still unknown what are computation overheads of AniPQO on industrial benchmark query templates. Finally, CG uses brute-force approach for producing plan diagram and CG-FPC costs all plans in POSP on every query location for reduction, which is brute-force approach; where AniPQO systematically explores selectivity space by making extremely less number of optimization calls for directly giving Reduced-POSP.

As AniPQO doesn't give guarantee of bounded cost-degradation on non-linear plan-cost-functions, it needs to be evaluated on industrial-strength optimizer to find how much cost-degradation bounds are violated by AniPQO. It also needs to evaluate AniPQO by comparing with CG-FPC in terms of reduced plan diagrams, reduction quality and computational overheads.

Need of a new error metric: In [6], AniPQO has been evaluated by using error metric *Maximum degradation of cost* over the selectivity space. This error metric measures whether the maximum degradation of cost over the selectivity space is greater than user-defined threshold, λ. If maximum degradation of cost is not greater than λ, then no query location is actually harmed as cost-degradation percentage at every location is atmost $\lambda\%$. But, if there are some query locations which are harmed, then this error metric is **not sufficient** for evaluation as it doesn't measure the portion of selectivity space that is harmed. Hence, a new error metric needs to be introduced called as *Harmed location percentage* formally defined in following section, which measures the percent of selectivity space, at which cost of inferred plan is not within small percent λ of cost of optimal plan. We can also replace error metric, maximum degradation of cost, with two other error metrics those are *Max Harm* and *Average Harm.* These metrics measure cost-degradation only at harmed locations, because we believe that cost-degradation at other locations, which is lesser than λ, is acceptable by user.

3.2 Problem definition

This work aims to evaluate AniPQO empirically on SQL Server 2008, an industrial-strength database engine, using error metrics "Harmed Locations" along with "Max Harm" and "Average Harm".

Given a query point $\boldsymbol{q}$ on grid of selectivity space $(\boldsymbol{SS})$, let $\boldsymbol{c_i(q)}$ be the cost of inferred plan, that is, the best plan in *AniPOSP* and $\boldsymbol{c_o(q)}$ be the cost of optimal plan at query point q. Let $\boldsymbol{\lambda}$ be user-defined threshold, that is, *maximum allowed cost-degradation percentage at any query point*, where $\lambda > 0$.

Now cost-degradation percentage, harmed location, harmed location percentage, max harm and average harm are defined formally as follows:

Cost-degradation percentage: The cost of inferred plan is greater than or equal to the cost of optimal plan and hence, there is some degradation of cost at query location q which is measured as *cost-degradation percentage* as follows

$$\boldsymbol{CDP(q) = \left[\left(\frac{c_i(q)}{c_o(q)} - 1\right) * 100\right]}$$

Harmed location percentage: It is the percentage of all query points in the grid of selectivity space, for which $[\boldsymbol{CDP(q) > \lambda}]$.

Max harm: Let $\boldsymbol{QH}$ be the set of all harmed query locations formally defined as

$$QH = \{q \mid q \in SS, CDP(q) > \lambda\}$$

Now, if there are some harmed locations, that is, harmed location percentage > 0, then max harm is defined as follows:

$$\boldsymbol{MaxHarm = \max_{q \in QH} (CDP(q) - \lambda)}$$

If harmed location percentage $= 0$, max harm is also 0.

Average harm: Same as max harm, it is defined when harmed location percentage > 0 as follows:

$$\boldsymbol{AverageHarm = \frac{\sum_{q \in QH} (CDP(q) - \lambda)}{\sum_{q \in QH} 1}}$$

If harmed location percentage $= 0$, average harm is also 0.

This work also aims to integrate AniPQO with Picasso to produce reduced plan diagrams directly, without generating plan diagrams beforehand.

Chapter 4

Analysing AniPQO on SQL Server 2008

For evaluation of AniPQO on SQL Server 2008, which is industrial-strength optimizer, we have developed an interface between AniPQO on SQL Server 2008, brought flexibility of providing an input to AniPQO and enhanced AniPQO to generate log files, for further analysis, while evaluating it. We have also integrated AniPQO with Picasso for generating reduced plan diagrams directly. In this chapter, these contributions are discussed in more details.

4.1 Interface with SQL Server

The implementation of AniPQO is in C++ and it is tightly coupled with Pyro's optimizer module. We have decoupled AniPQO's module and developed a new interface that operates as a middleware between AniPQO and SQL Server 2008. Interface is developed in the perspective of general use, that is, any other commercial optimizer can be used in the place of SQL Server 2008 with minor changes in the implementation. Now, AniPQO can also be evaluated on any optimizer other than SQL Server 2008. We have used *Open Database Connectivity* (ODBC) driver for developing this interface.

4.2 Flexible input

Current implementation works on the integer parameter space, that is, consider a parametric query with 2 varying predicates from different relations. Range of attribute in first predicate is from 1 to 1000 and range of second is from 1 to 500 (integer values), then the parameter space is the grid of query points of resolution 1000×500. This parameter space is generated by Pyro, which uses metadata of database. This parameter space is fixed and can not be set by user. When range of varying predicates is large, then parameter space will also have very large number of query points.

We have brought the flexibility in providing the input, so that AniPQO now works on a grid of selectivity space of *any resolution* and these selectivity constants for each dimension in query template can be chosen by the user as per the required distribution of query instances. We have used selectivity constants generated directly by Picasso. In experiments, we have tested AniPQO on grid of resolution 300×300 and $100 \times 100 \times 100$ for various query templates with both uniform and exponential distributions [7] for λ; like 1%, 5%, 10% and 20%.

4.3 Visualization of reduced plan diagram given by AniPQO

In [6], they have shown that quality of plans is very good, that is, degradation of cost is not significantly larger than λ. But, this doesn't give any idea about geometries of optimality regions of plans in $AniPOSP$. Hence, it becomes very essential to visualize the results. For this reason, we have done visualization of AniPQO's results with the help of Picasso, version 2.1, using two different designs. One is the loose integration, that is, production of reduced plan diagram after evaluating AniPQO and another is tighter integration with Picasso, that is, production of *on the fly* reduced plan diagram using Picasso with the help of AniPQO.

4.3.1 Reduced plan diagram after evaluating AniPQO

In this design, we generate log files in the *error computation phase*, explained in more details in Subsection 5.1, for analysis and for production of reduced plan diagrams. These log files store various fields for every query location; including optimal plan number, optimal plan's cost, inferred plan number, inferred plan's cost and cost-degradation percentage. We have used this information to produce reduced plan diagrams with the help of Picasso.

Picasso produces plan diagram after two phases. In first phase, it makes optimization calls to the query optimizer for each query location to get various information including optimal plan and its cost. It stores all this information in the database to avoid redoing of this work for production of the same plan diagram in the future. In the second phase, it reads back all that stored information from the database and generates plan diagram by systematically assigning colors to optimality regions [7].

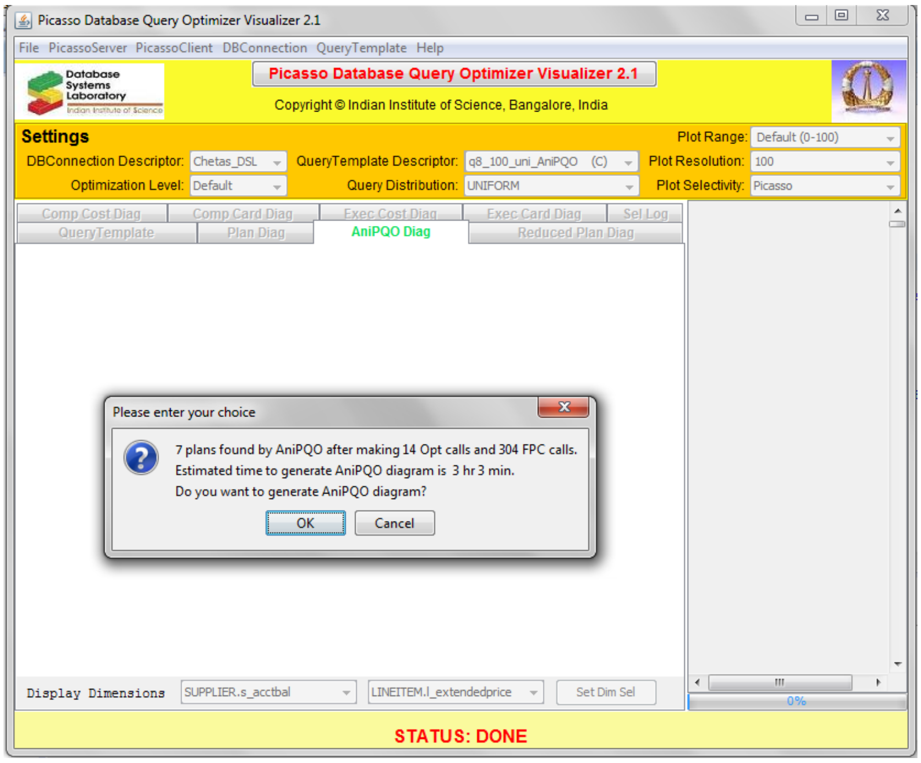
We have generated reduced plan diagrams by exploiting the second phase. We modified Picasso's implementation to read plan numbers and their costs from these log files instead of reading back from the database and rest of the procedure of generating plan digram by assigning colors to the optimality regions is the same. We do not store information related to currently generated reduced plan diagram in the database, as we already have this information locally in the log files.

4.3.2 Production of on the fly reduced plan diagrams

In this design, we have enhanced implementation of Picasso to directly produce reduced plan diagram with the help AniPQO. We have introduced new panel, called as *AniPQO diagram*, added next to the *Plan diagram* panel to display AniPQO's reduced plan diagram. Users can now select this panel after loading query template in the *Query template* panel. Picasso asks user an "optimality threshold" which is given as a input, λ, to AniPQO. Picasso then runs AniPQO, gets *AniPOSP* and estimates time for generating reduced plan diagram. This estimated time along with the information including number of plans in *AniPOSP*, number of optimization calls and FPC calls made by AniPQO is prompted to user and reduced plan diagram is generated after confirmation from user.

Like plan diagram, reduced plan diagram is also produced after two phases. In first phase, each plan in *AniPOSP* is costed at every query location to find best plan, that is, inferred plan. An information including inferred plan number and inferred cost at every query location is stored in the database to reproduce the same reduced plan diagram in the future. In the second phase, all that stored information is read back from the database to generate reduced plan diagram by assigning colors to the optimality regions.

Figure 4.1 and Figure 4.2 exemplify a reduced plan diagram generated by tight integration of AniPQO with Picasso. Figure 4.1 shows that after giving QT8 and $\lambda = 1\%$ as a input, Picasso runs AniPQO and shows estimated generation time of reduced plan diagram. After getting confirmation from user, Picasso generates and displays reduced plan diagram as shown in Figure 4.2.


Picasso Database Query Optimizer Visualizer 2.1
File PicassoServer PicassoClient DBConnection QueryTemplate Help
Database Systems Laboratory
Indian Institute of Science
Picasso Database Query Optimizer Visualizer 2.1
Copyright © Indian Institute of Science, Bangalore, India
Settings
Comp Cost Diag
Comp Card Diag
Exec Cost Diag
Exec Card Diag
Sel Log
QueryTemplate
Plan Diag
AniPQO Diag
Reduced Plan Diag
OK
Cancel
Set Dim Sel
0%

Plot Range: Default (0-100)
DBConnection Descriptor: Chetas_DSL
QueryTemplate Descriptor: q8_100_uni_AniPQO (C)
Plot Resolution: 100
Optimization Level: Default
Query Distribution: UNIFORM
Plot Selectivity: Picasso

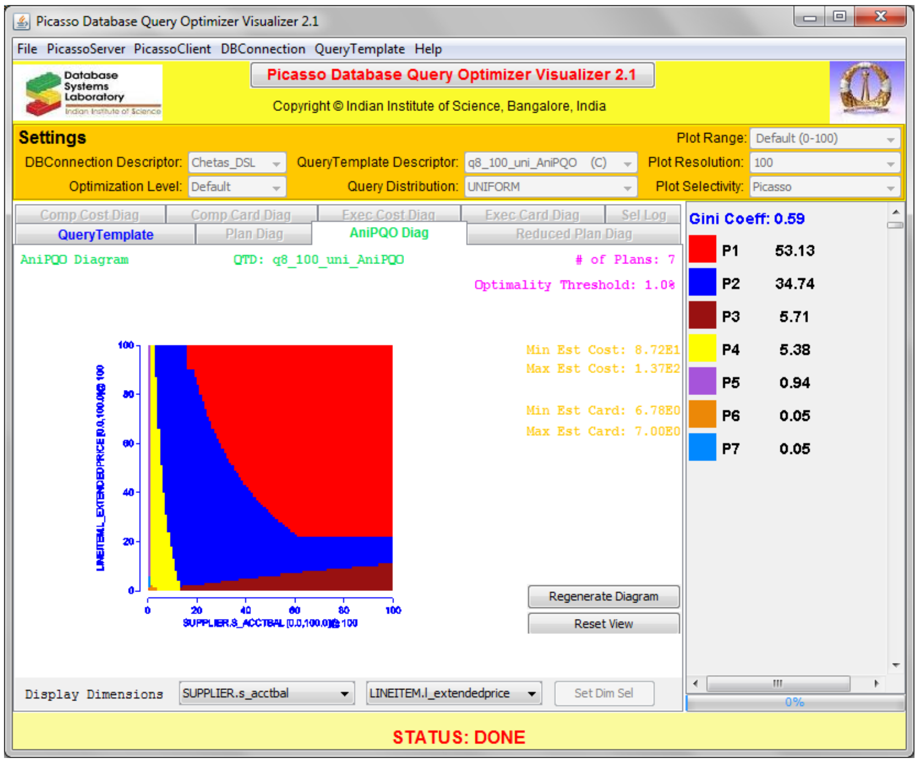
Please enter your choice

7 plans found by AniPQO after making 14 Opt calls and 304 FPC calls.
Estimated time to generate AniPQO diagram is 3 hr 3 min.
Do you want to generate AniPQO diagram?

Display Dimensions SUPPLIER.s_acctbal LINEITEM.l_extendedprice

STATUS: DONE

Figure 4.1: Estimated time for producing reduced plan diagram of QT8 with $\lambda = 1\%$


Picasso Database Query Optimizer Visualizer 2.1
File PicassoServer PicassoClient DBConnection QueryTemplate Help
Database Systems Laboratory
Indian Institute of Science
Picasso Database Query Optimizer Visualizer 2.1
Copyright © Indian Institute of Science, Bangalore, India
Settings
Comp Cost Diag
Comp Card Diag
Exec Cost Diag
Exec Card Diag
Sel Log
QueryTemplate
Plan Diag
AniPQO Diag
Reduced Plan Diag
LINEITEM.L_EXTENDEDPRICE [0.0,100.0]% 100
SUPPLIER.S_ACCTBAL [0.0,100.0]% 100
0
20
40
60
80
100
Regenerate Diagram
Reset View
Set Dim Sel
0%

Plot Range: Default (0-100)
DBConnection Descriptor: Chetas_DSL
QueryTemplate Descriptor: q8_100_uni_AniPQO (C)
Plot Resolution: 100
Optimization Level: Default
Query Distribution: UNIFORM
Plot Selectivity: Picasso

AniPQO Diagram
QTD: q8_100_uni_AniPQO
# of Plans: 7
Optimality Threshold: 1.0%

Min Est Cost: 8.72E1
Max Est Cost: 1.37E2

Min Est Card: 6.78E0
Max Est Card: 7.00E0

Gini Coeff: 0.59

🟥	P1	53.13
🟦	P2	34.74
🟫	P3	5.71
🟨	P4	5.38
🟪	P5	0.94
🟧	P6	0.05
🟦	P7	0.05

Display Dimensions SUPPLIER.s_acctbal LINEITEM.l_extendedprice

STATUS: DONE

Figure 4.2: Reduced plan diagram of QT8 produced by using AniPQO, # plans 7

Chapter 5

Evaluation Scenarios

We have evaluated AniPQO to see how much cost-degradation bounds are violated and also compared with CG-FPC in terms of reduced plan diagrams, reduction quality and computational overheads. We have also checked scalability of AniPQO with the dimension. In this chapter, evaluation scenarios are discussed.

In SQL Server 2008, according to system workload, cost of execution plan varies with available memory, to get consistency in experimental results, we have fixed memory parameters "minimum server memory" and "maximum server memory" to be 1792 MB (1.75 GB) and "minimum memory per query" to be 200 MB. We have used *query hinting* feature of SQL Server 2008 by using query option "USE PLAN" as a FPC feature. But this feature is very limited, that is, SQL Server's optimizer may or may not use suggested plan as it is and we may get inflated cost of foreign plan. For example, suppose we got a plan p after optimizing one query location, while costing the same plan p at same query location, optimizer may alter this plan which results into inflated cost. We call this particular FPC-error as "self-FPC error". We have also used query option "MAXDOP 1" for avoiding parallelism to avoid FPC errors as much as possible.

In experiments, we enhanced implementation to store application trace, error log and MATLAB script in files. Application trace shows how AniPQO worked on a given query. Error log is created at the time of error computation. This log has information for each query point like cost-degradation percentage. Fields like optimal plan, optimal plan's cost, inferred plan, inferred plan's cost for each harmed location is stored in error log for analysis. While we may get some FPC error while costing inferred plan at query point, this is also logged in error log file. To visualize harmed locations, we enhanced AniPQO to generate MATLAB scripts to plot harmed locations. We assigned color codes to query points to identify error ranges, that is, cost-degradation percentage.

- Yellow: self FPC-error
- Green: λ to $\lambda + 10$
- Blue: $\lambda + 10$ to $\lambda + 20$
- Red: $\lambda + 20$ to $\lambda + 30$
- Black: greater than $\lambda + 30$

5.1 Evaluation on TPC-H and TPC-DS decision support benchmarks

We have evaluated AniPQO on TPC-H benchmark database of 1 GB scale with two different physical designs of our database engine, that is, **PrimaryKey(PK)** and **AllIndex(AI)**. PK represents the physical design, wherein a clustered index is created on primary key of each relation. AI, on the other hand, represents an "index-rich" situation wherein (single-column) indices are available on all query-related schema attributes. For evaluation, we have considered physical design AI, which was not considered in [6], because AI has richer schema in which query optimizer has many options to choose indexes in an execution plan. We have also evaluated AniPQO on TPC-DS [8] benchmark database of 100 GB scale with the physical design PK.

We have done evaluation of AniPQO on query templates based on benchmark queries as follows:

1. **Optimization:** We find optimal plans and their costs at every query location to find actual POSP. This is the brute-force way of finding POSP. This phase is required because cost of optimal plan is required to calculate cost-degradation for calculation of harmed location percentage

2. **AniPQO:** We run AniPQO on query template using various λ values, including $1\%, 5\%, 10\%$ and 20%, to get $AniPOSP$. In [6], if some plan in $AniPOSP$ is already co-optimal with new plan found by optimizing unoptimized vertex, this new plan is not used for further decomposition instead of discarding it and still added in $AniPOSP$ to improve quality of plans $AniPOSP$ for ensuring lesser harm. We have not added these new plans if some co-optimal plans already exist in $AniPOSP$, to ensure anorexic level.

3. **Error computation:** We find best plan in $AniPOSP$ at each query location by costing every plan in $AniPOSP$, to get cost of inferred plan to calculate errors including harmed

Query Template	**λ (%)**	**# plans**		**Harmed Locations (%)**	**Max Harm (%)**	**Avg Harm (%)**	**Optimizer calls**	**FPC calls**
		POSP	**AniPOSP**					
QT3 (exp)	1	33	7	0.14	7.73	2.46	14	407
	5		5	0.05	3.73	0.89	11	143
	10		4	0	0	0	10	93
	20		3	1.38	60.15	14.30	8	74
QT3 (uni)	1	14	6	1.46	2.82	1.67	8	125
	5		5	0	0	0	7	91
	10		3	0	0	0	6	42
	20		3	0	0	0	6	31
QT4 (exp)	1	9	4	0	0	0	9	151
	5		4	0	0	0	9	103
	10		3	0	0	0	8	80
	20		3	0	0	0	8	80
QT4 (uni)	1	5	3	0	0	0	6	48
	5		2	0	0	0	6	32
	10		2	0	0	0	6	28
	20		2	0	0	0	6	26
QT8 (exp)	1	138	9	38.83	7.02	1.68	15	290
	5		3	26.81	6.04	2.42	7	31
	10		3	1.46	1.04	0.39	7	24
	20		2	2.08	6.19	2.76	5	11
QT8 (uni)	1	68	9	39.35	7.23	2.86	20	666
	5		5	10.90	3.23	0.96	7	52
	10		3	1.49	1.07	0.37	5	12
	20		3	0	0	0	5	10

Table 5.1: Evaluation results of TPC-H physical design PK with resolution 300 $\times$ 300

location percentage, max harm and average harm. We also generate log files for further analysis.

5.2 Evaluation results on TPC-H(PK)

Table 5.1 shows evaluation results on TPC-H benchmark with physical design, PK, with resolution 300 $\times$ 300. We have evaluated query templates QT3, QT4 and QT8, which found to be less FPC error prone. Table shows that cardinality of POSP for exponential distribution is greater than uniform distribution, because more number of optimal plans are found at lower selectivities. As λ increases, cardinality of *AniPOSP* exponentially decreases and becomes constant for 10% and 20%. Table 5.1 also shows harmed location percentage. Harmed location percentage for QT4 is 0. Hence, *AniPOSP* for QT4 can be considered as Reduced-POSP. Harmed location percentage for QT8 with λ = 1% is very high, but QT8 has more complex plan diagram compared to other query templates, having dense and large number of plans in POSP. We have visualized these harmed locations of QT8 with uniform distribution and $\lambda = 1\%$ shown in Figure 5.1. Figure shows harmed locations percentage is significantly high, but all harmed locations have cost-degradation percentage lesser than $\lambda + 10$ and even some of them are due to self-FPC error; where optimal plan is present in *AniPOSP* and AniPQO also inferred same plan as inferred plan, but the cost of same plan after doing FPC got inflated and query location became harmed location.

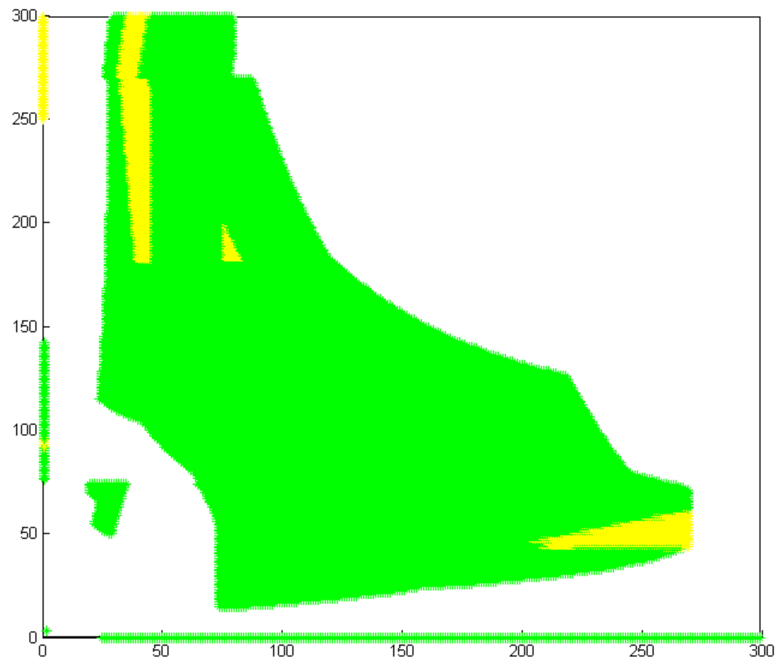


Figure 5.1: Harmed locations for QT8 with uniform distribution and $\lambda = 1\%$

Max harm indicates extra cost-degradation percentage beyond acceptable cost-degradation percentage, that is, λ. Max harm in all cases is **within 10%**, except QT3 with exponential distribution. In this exceptional case, AniPQO missed one important plan which caused very high cost-degradation at certain portion of selectivity space, this is because it is a heuristic technique which assumes traversal intersection between cost functions [6], which may not hold always. Although results show significant harmed location percentage, average harm is extremely low, that is, **within or around 3%**, except QT3 with exponential distribution, which indicates that most of the harmed locations are having extra cost-degradation lesser than 3%.

The table shows that the number of optimizer calls made by AniPQO are extremely less as compared to number of query points in the selectivity space, that is, $300 * 300$. Even number of FPC calls made by AniPQO are also very less. This shows that AniPQO identified *AniPOSP* without exploring selectivity space extensively, like brute force approach. Hence, although

Query Template	λ (%)	# plans		Harmed Locations (%)	Max Harm (%)	Avg Harm (%)	Optimizer calls	FPC calls
		POSP	AniPOSP					
QT7 (exp)	1	43	18	12.83	11.97	5.18	31	503
	5		12	1.07	7.97	5.19	24	463
	10		10	0.01	2.97	6.10	23	940
	20		9	0	0	4.53	20	651
QT7 (uni)	1	32	10	7.63	7.78	4.84	20	458
	5		9	1.88	3.78	0.89	19	596
	10		8	0.90	3.07	1.27	16	485
	20		6	0.86	37.01	13.62	13	572
QT8 (exp)	1	38	11	0.03	1.20	8.99	22	330
	5		9	0	0	0	18	262
	10		9	0	0	0	18	240
	20		8	0	0	0	13	168
QT8 (uni)	1	14	6	0	0	0	15	270
	5		6	0	0	0	14	330
	10		5	0	0	0	10	191
	20		5	0	0	0	9	160
QT9 (exp)	1	50	12	0.16	32.51	8.53	24	369
	5		12	0.06	9.51	2.78	26	1008
	10		9	0.01	9.95	3.05	19	578
	20		7	0	0	0	15	404
QT9 (uni)	1	19	5	32.12	7.71	2.94	8	249
	5		5	9.89	3.71	1.32	8	393
	10		4	0	0	0	8	160
	20		3	0	0	0	7	53

Table 5.2: Evaluation results of TPC-H physical design AI with resolution 300 × 300

AniPQO has given significant harmed location percentage with max harm, it should also be noted that *AniPQO has performed very efficiently with extremely less computation*, because FPC calls are very cheap as compared to optimization calls. As λ increases, number of FPC calls also decreases exponentially.

5.3 Evaluation results on TPC-H(AI)

We have also done evaluation on TPC-H benchmark with physical design, AI, in which, query optimizer has many options to choose indexes in an execution plan, which causes the cost of the optimal plan to be very low. Now even if absolute cost-difference between foreign plan and optimal plan is not significantly large, relative cost-difference in percentage may be very large. Due to this, AniPQO may perform badly because it finds centre point of hyper-rectangle a lot of times during its execution, where optimal plans in hyper-rectangle are equi-cost at this centre point, that is, *relative cost-difference* of optimal plans is not larger than λ. This can be cleared with following example. In case of PK, if cost of optimal plan is 50 and cost of inferred plan is 55, relative cost-difference 10%. But in case of AI, if optimal cost is 5 and cost of inferred cost is 10, relative cost-difference is 100%. Even though absolute cost-difference is 5 in both cases, relative cost-difference looks very high in case of AI. For this reason, we have not considered query points as harmed query locations where absolute costs of optimal plan and inferred plan are *lesser than 10*.

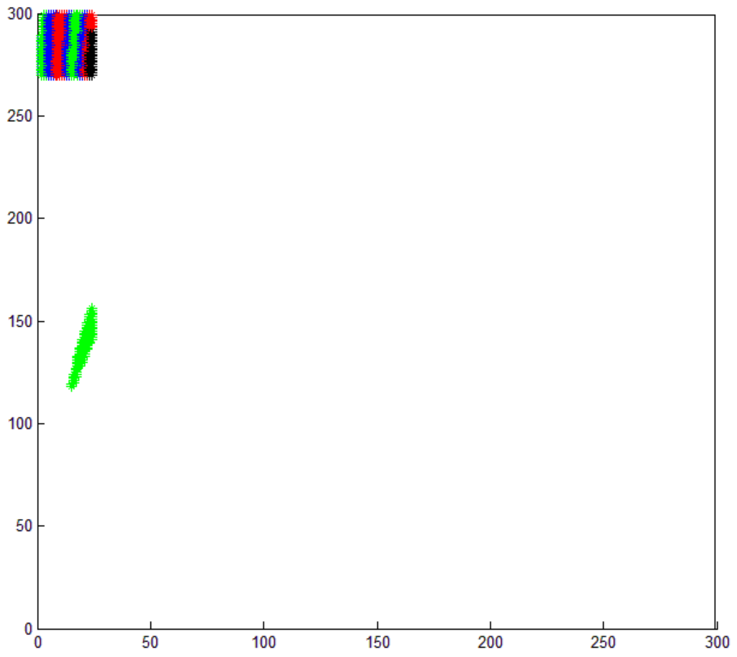


Figure 5.2: Harmed locations for QT7 with uniform distribution and $\lambda = 20\%$

Table 5.2 shows evaluation results for AI with resolution 300×300. We have evaluated query templates QT7, QT8 and QT9. Table shows that FPC calls does not decrease as λ increases. Reason of this interesting observation lies in explanation given in previous paragraph. This happens for small λ, like 1% and 5%, where centre points of some hyper-rectangles may not be found during execution of AniPQO, because it considers cost-difference *relatively* and as costs of plans happen to be very small in AI, equi-cost points may not be found. This affects further execution and some important optimal plans may be missed by AniPQO. Hence, Table 5.2 shows that harmed location percentage for small λ is significantly high as compared to large λ, because AniPQO missed some optimal plans.

Table 5.2 shows that max harm in AI is very high, for some cases, as compared to PK. This is again because of relative cost-difference. As costs are very low in AI as compared to PK, relative cost-difference will be very high in case of AI. Table shows that for all query templates, except QT7 with uniform distribution and QT9 with exponential distribution, average harm is

Query Template	λ (%)	# plans		Harmed Locations (%)	Max Harm (%)	Avg Harm (%)	Optimizer calls	FPC calls
		POSP	AniPOSP					
DSQT18 (exp)	1	235	8	0.20	12.15	2.82	15	186
	5		4	0.60	8.15	1.85	8	45
	10		4	0.01	3.15	1.51	7	39
	20		3	0	0	0	7	32
DSQT18 (uni)	1	66	11	0.60	7.60	1.86	18	157
	5		7	0.11	3.76	1.25	13	150
	10		4	0	0	0	8	37
	20		3	0	0	0	6	20
DSQT19 (exp)	1	48	5	0.41	8.15	2.03	12	115
	5		5	0.02	4.15	1.23	10	66
	10		4	0.47	8.23	4.23	7	43
	20		4	0	0	0	7	39
DSQT19 (uni)	1	44	6	0.46	2.04	0.72	13	186
	5		5	0	0	0	8	52
	10		5	0	0	0	8	50
	20		4	0	0	0	7	37
DSQT61 (exp)	1	89	12	3.37	20.15	7.53	22	584
	5		11	0	0	0	15	597
	10		10	0	0	0	13	619
	20		8	0.01	1.14	0.46	12	422
DSQT61 (uni)	1	61	10	2.81	2.75	1.12	15	464
	5		8	0	0	0	13	473
	10		6	4.45	1.10	0.58	10	104
	20		2	39.51	11.64	4.45	5	18

Table 5.3: Evaluation results of TPC-DS physical design PK with resolution 100 × 100

not significantly large, which indicates that degradation of cost for most of the harmed locations is within or around average harm. Even in exceptional case of QT7 with uniform distribution, harmed location percentage is very less which is shown in Figure 5.2[1]. Figure shows that some query locations are severely harmed harmed shown as red and black markers, but those are in very less number as compared to green and blue markers.

5.4 Evaluation results on TPC-DS

We have also done evaluation of AniPQO on TPC-DS benchmark database, scale 100 GB, of physical design PK using query templates DSQT18, DSQT19, DSQT61; results shown in Table 5.3. Similar to Table 5.1, cardinality of *AniPOSP* also decrease exponentially as λ increases. Harmed location percentage is not significant for all query templates, except DSQT61 with uniform distribution and $\lambda = 20\%$ where AniPQO missed some optimal plans due to the

[1]The figure in this report should ideally be viewed from color copy to clearly register the feature

assumption of traversal intersection between cost functions. For $\lambda = 1\%$, max harm looks significantly large for some query templates, but average harm shows that most of the harmed locations have extra cost-degradation beyond $\lambda\%$ *within and around* 5%, except DSQT61 with exponential distribution and $\lambda = 1\%$. Table shows that AniPQO explored significantly less portion of selectivity space as compared to brute force approach giving very less harmed location percentage for most of the cases. Harmed locations of DSQT61 with uniform distribution and $\lambda = 20\%$ are shown in Figure 5.3. Figure shows that although harmed location percentage is significantly high, most of the harmed locations have cost-degradation percentage within $\lambda + 10$.

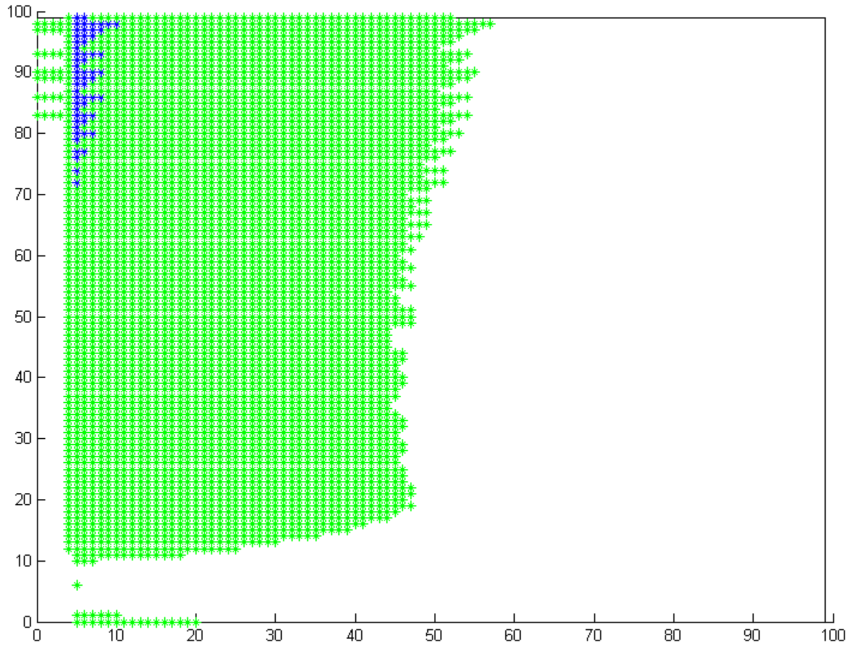


Figure 5.3: Harmed locations for DSQT61 with uniform distribution and $\lambda = 20\%$

5.5 Evaluation results on 3-D query templates

We have also done evaluation of AniPQO on 3-D query templates on TPC-H benchmark of physical design PK. Table 5.4 shows results of evaluation. Table shows that cardinality of POSP also increases drastically as dimension of query template increases. Table shows that AniPQO still gives cardinality of *AniPOSP* to **anorexic level** for even 3-D query templates having large number of plans in POSP. Similar to Table 5.1, cardinality of *AniPOSP* and number FPC calls also decrease exponentially as λ increases. Harmed locations for QT8 with $\lambda = 10\%$ and 20% is 0, which shows that AniPQO returns Reduced-POSP with extremely less computation. For QT8 with $\lambda = 1\%$, max harm looks significantly large, but average

Query Template	λ (%)	# plans		Harmed Locations (%)	Max Harm (%)	Avg Harm (%)	Optimizer calls	FPC calls
		POSP	AniPOSP					
QT8(uni)	1	268	13	32.23	7.31	2.49	37	2186
	5		7	5.22	3.31	0.86	30	1354
	10		6	0	0	0	26	920
	20		5	0	0	0	14	100
QT9(uni)	10	421	9	6.86	13.86	4.09	23	365
	20		5	1.14	12.57	2.29	13	134

Table 5.4: Evaluation results of TPC-H physical design PK with resolution $100 \times 100 \times 100$

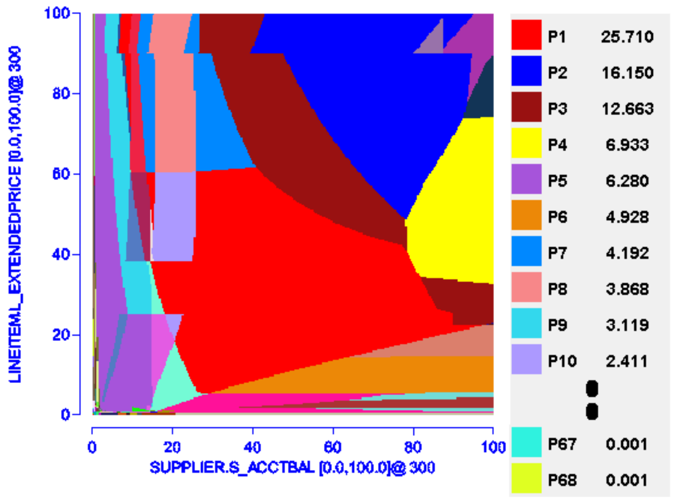
harm shows that most of the harmed locations have extra cost-degradation percentage beyond λ *within and around* 2.49%. It shows that Max harm for QT9 is very high as compared to QT8, but again average harm shows that cost-degradation is not high for most of the harmed locations. In spite of significantly very high max harm for QT9, AniPQO made extremely less number of FPC calls as compared to the number of query points, that is, $100 * 100 * 100$; which shows that AniPQO still gives significantly less max harm and average harm by exploring very small portion of selectivity space, which is not even 0.01% of total selectivity space.

5.6 Comparison AniPQO and CG-FPC

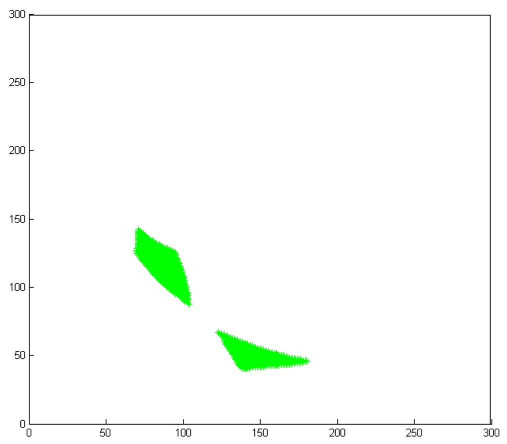
As mentioned before, CG-FPC is the anorexic reduction technique, variant of CG, which utilizes FPC feature. CG-FPC gives better reduction quality than CG. But, CG-FPC costs every plan in the POSP at every query locations, which makes it *extremely time-consuming* with enormous computations. On the other hand, reduced plan diagrams can also be generated by using AniPQO. As AniPQO *directly* gives Reduced-POSP (heuristically), reduced plan diagrams can directly be generated by costing each plan in *AniPOSP*, to find the best plan, at each query location. We have compared AniPQO and CG-FPC using reduced plan diagrams, reduction quality and computational overheads.

5.6.1 Comparison using reduced plan diagrams

We compare reduced plan diagrams produced by using AniPQO and CG-FPC on QT8, on PK design, with uniform distribution and $\lambda = 10\%$. The reduced plan diagram produced by using AniPQO is shown in Figure 5.5a and Figure 5.5b shows reduced plan diagram, reduced by using CG-FPC. The number of plans in *AniPOSP* are just 3 (at anorexic level) as compared to size of POSP, that is, 68 and number of plans after reduction using CG-FPC are only 2. Figures show that reduced plan diagrams using AniPQO and CG-FPC **look quite similar**; AniPQO has just one extra plan in *AniPOSP*, but it covers only 0.081% of the selectivity space and rest

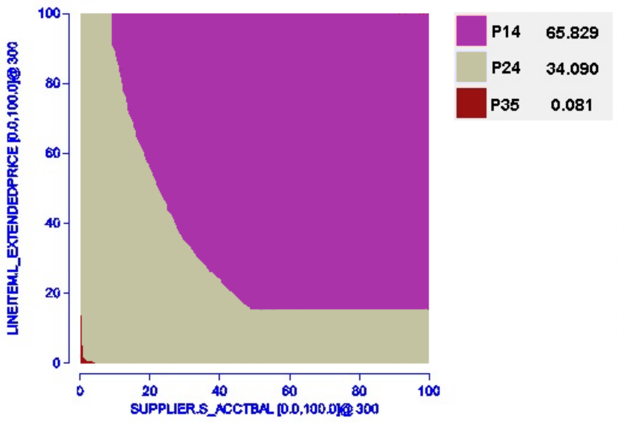


(a) QT8 plan diagram (# plans 68)

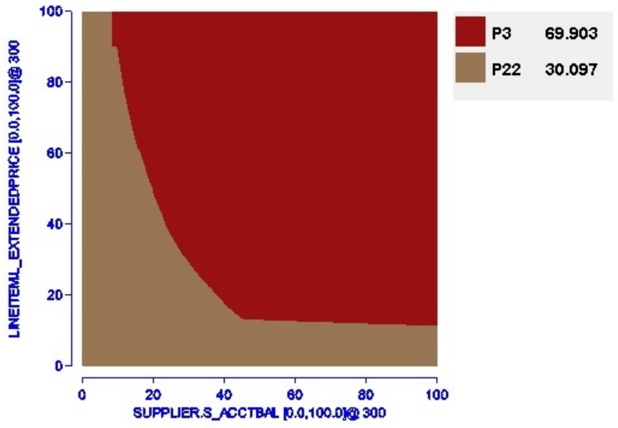


(b) Harmed locations of QT8 with $\lambda = 10\%$

Figure 5.4: Plan diagram and harmed locations of QT8



(a) AniPQO ($\lambda = 10\%$, # plans 3)

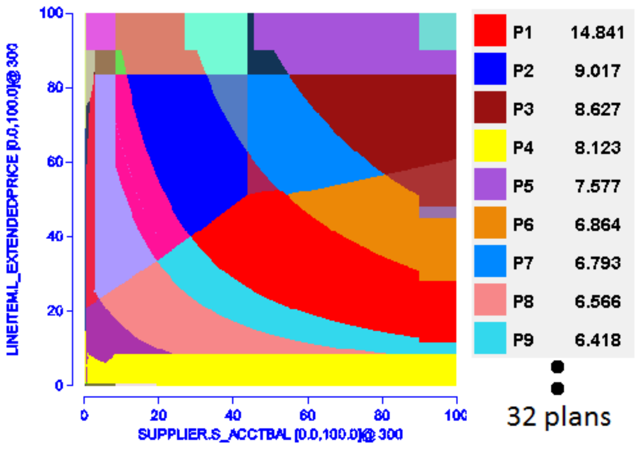


(b) CG-FPC ($\lambda = 10\%$, # plans 2)

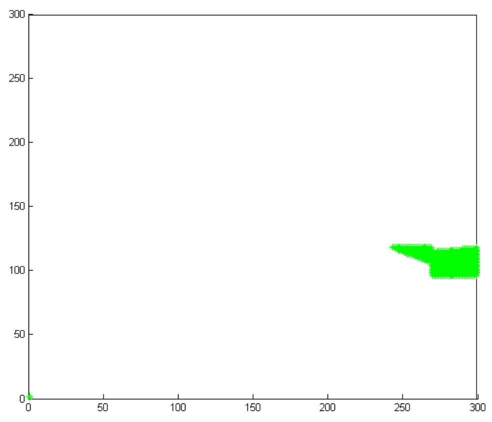
Figure 5.5: Reduced plan diagrams of QT8 given by AniPQO and CG-FPC

of the optimality regions are nearly similar. Figure 5.4b shows harmed locations of QT8 given by AniPQO with $\lambda = 10\%$. The harmed location percentage looks slightly high, but Table 5.1 shows that max harm is just about **1**% which is not significant and it is still acceptable. The rest of the selectivity space is not harmed which shows that AniPQO has violated bounded cost-degradation for small portion of the selectivity space.

We further compare reduced plan diagrams of QT7, on AI design, with uniform distribution and $\lambda = 10\%$. The original plan diagram of QT7 with uniform distribution is shown in Figure 5.6a, plan diagram has 32 plans. The reduced plan diagrams produced by using AniPQO and CG-FPC are shown in Figure 5.7a and Figure 5.7b respectively. The number of plans in

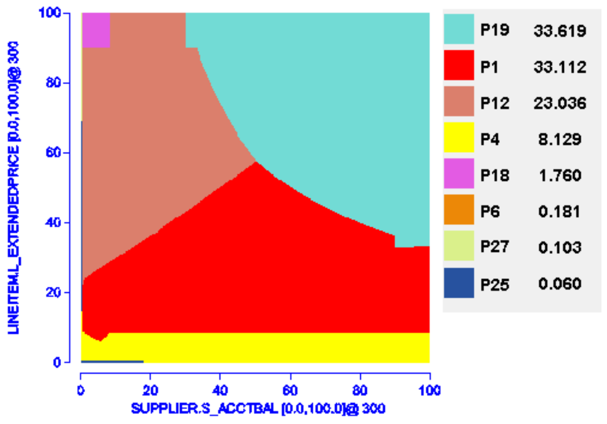


(a) QT7 plan diagram (# plans 32)

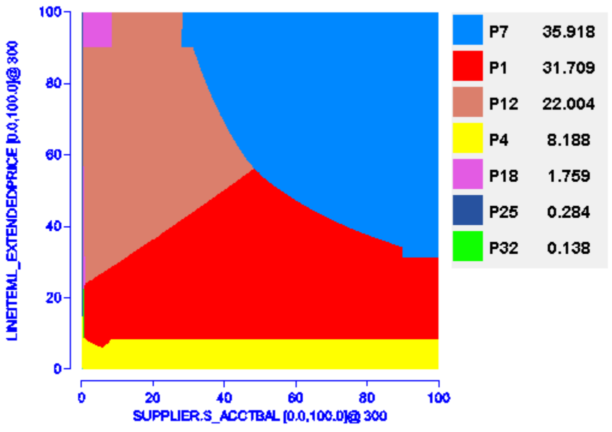


(b) Harmed locations of QT7 with $\lambda = 10\%$

Figure 5.6: Plan diagram and harmed locations of QT7



(a) AniPQO ($\lambda = 10\%$, # plans 8)



(b) CG-FPC ($\lambda = 10\%$, # plans 7)

Figure 5.7: Reduced plan diagrams of QT7 given by AniPQO and CG-FPC

$AniPOSP$ are just 8 (at anorexic level) and number of plans after reduction using CG-FPC are only 7. Figures show that reduced plan diagrams using AniPQO and CG-FPC **look almost same**; AniPQO has just one extra plan in $AniPOSP$, but it covers only 0.060% of the selectivity space and rest of the optimality regions are nearly same. We are getting such a close match because in [6], a technique of *approximate indexing* (by using $AniPOSP$) is suggested to quickly infer optimal plan from $AniPOSP$ at the run-time and this technique can be used in a scenario when cardinality of $AniPOSP$ is very high. As we have found that cardinality of $AniPOSP$ is at the anorexic level, we are finding the best plan from $AniPOSP$ (by costing each plan) while producing reduced plan diagram.

Figure 5.6b shows harmed locations of QT7 given by AniPQO with $\lambda = 10\%$. The harmed location percentage looks slightly high, but Table 5.2 shows that max harm is just about **3**% which is not significantly high and it is still acceptable. The rest of the selectivity space is not harmed which shows that AniPQO has *not violated* bounded cost-degradation for **99**% of the total selectivity space.

λ (%)	**AniPQO/ CG-FPC**	**QT-7**		**QT-8**		**QT-9**	
		Exp	**Uni**	**Exp**	**Uni**	**Exp**	**Uni**
1	AniPQO	18	12	11	6	12	5
	CG-FPC	27	16	15	6	18	7
5	AniPQO	12	9	9	6	12	5
	CG-FPC	17	10	11	5	7	5
10	AniPQO	10	8	9	5	9	4
	CG-FPC	12	7	10	4	9	4
20	AniPQO	9	6	8	5	7	3
	CG-FPC	10	6	8	3	6	3

Table 5.5: Comparing reduction quality of AniPQO with CG-FPC

5.6.2 Comparison using reduction qualities

We now compare AniPQO and CG-FPC in terms cardinality of *AniPOSP* and Reduced-POSP given by CG-FPC shown in Table 5.5. Table shows cardinality of AniPOSP and Reduced-POSP given by CG-FPC for query templates on AI physical design. As explained in Section 5.3, AniPQO may miss some optimal plans for small λ, like 1% and 5%. Table 5.5 shows the same, where AniPQO has given less number of plans as compared to CG-FPC for $\lambda = 1\%$ and 5%, except QT9 with exponential distribution and $\lambda = 5\%$. In this exceptional case, AniPQO added more number of plans in *AniPOSP* by exploring selectivity space significantly more than other query templates with the same distribution and λ. Table 5.2 shows that harmed location percentage, max harm are also very less and number of FPC calls made by AniPQO are significantly high. But in case of λ, like 10% and 20%, cardinalities are almost equal. This shows that for $\lambda = 10\%$ and 20%, AniPQO gives the Reduced-POSP; because max harm, average harm and harmed location percentage is not significantly large.

Reduction quality of AniPQO and that of CG-FPC is almost same for $\lambda = 10\%$ and 20%. Hence, *AniPOSP* can be considered as Reduced-POSP for these λ values.

5.6.3 Comparison using computational overheads

We have shown reduced plan diagrams of QT8 with uniform distribution, 300×300 resolution and $\lambda = 10\%$; produced by using AniPQO and CG-FPC in Figure 5.5a and Figure 5.5b respectively. AniPQO returned $AniPOSP$ having 3 plans. After reducing the original plan diagram with 68 plans, CG-FPC reduced to only 2 plans, where $AniPOSP$ also has only 3 plans. Though reduced plan diagrams of AniPQO and CG-FPC looks very similar, plans in $AniPOSP$ and CG-FPC are not same. In case of CG-FPC, degradation of cost is at most 10%. But, in case of AniPQO, some harmed locations have found with max harm as 1%, that is, 1% more than the acceptable degradation, which is not significantly large.

	CG-FPC	**AniPQO**	**CG**
Optimization Calls	300*300	5	300*300
FPC Calls	68*300*300	12+(3*300*300)	-

We subsequently compare computational overheads in both ways. AniPQO made 5 optimization calls and 12 FPC calls in its execution to find out $AniPOSP$. Subsequently, it made $3 * 300 * 300$ FPC calls to produce reduced plan diagram. On the other side, CG-FPC needs plan diagram beforehand, which is produced after making $300 * 300$ optimization calls. Then CG-FPC costed all 68 plans on all $300 * 300$ query locations for reduction. As compared to CG-FPC, AniPQO produced plan diagram with extremely less computation, because optimization calls are very costly as compared to FPC calls. If we consider CG, it doesn't make FPC calls, but it still needs plan diagram to be produced for reduction, where AniPQO doesn't need it. We compare number of FPC calls made in both reduction techniques as follows.

Let cardinality of POSP and $AniPOSP$ be $\boldsymbol{P}$ and $\boldsymbol{A}$ respectively. Let resolution be $\boldsymbol{R}$ and dimension be $\boldsymbol{D}$. Number of FPC calls made by AniPQO and CG-FPC are $(\boldsymbol{A} * \boldsymbol{R}^{\boldsymbol{D}})$ and $(\boldsymbol{P} * \boldsymbol{R}^{\boldsymbol{D}})$ respectively. In this chapter, we have shown results of evaluation on various query templates, which shows that the cardinality of $AniPOSP$ is very small, at anorexic level, as compared to the cardinality of POSP, that is, $\boldsymbol{A} \ll \boldsymbol{P}$. This shows that FPC calls made by AniPQO for any query template are extremely small as compared to CG-FPC. We have compared AniPQO with CG-FPC on many TPC-H query templates on both physical designs. With a lot of evidence we have a strong claim that *AniPQO is as fast as CG and as reductive as CG-FPC.*

5.7 Scalability with dimension

Time complexity of producing plan diagram, that is, brute-force approach is exponential with respect to the dimension. Hence, production of plan diagram becomes impractical after 4 dimensions even for resolution 100. But while increasing dimension, AniPQO can still produce *AniPOSP* on such higher dimension making extremely less number of optimization calls as compared to brute-force approach. Hence, it becomes essential to check how AniPQO scales with the dimension. We have tested AniPQO on QT8 and QT9 with uniform and exponential distribution respectively. We have found that the number of optimization calls and FPC calls increase exponentially, but this rate of increase is significantly lesser than that of brute-force approach, that is, AniPQO still explores very less portion of selectivity space on higher dimensions.

We have run AniPQO on query templates QT8 and QT9 with resolution 100, results are shown in the Table 5.6. Table shows the increase of number of optimization calls and FPC calls made by AniPQO to explore selectivity space for returning *AniPOSP*. As dimension increases, optimization and FPC calls also increase exponentially. This exponential growth is expected, because vertices and edges of the decomposition space also increase exponentially.

Query Template	**Dimension**	**AniPOSP**	**Opt calls**	**FPC calls**
QT8 (uni)	1	2	2	2
	2	5	9	97
	3	10	32	1003
	4	14	51	5639
	5	19	143	19354
QT9 (exp)	1	4	6	32
	2	7	14	141
	3	12	28	1012
	4	23	61	9207
	5	25	126	17322

Table 5.6: Scalability with dimension, resolution 100 and $\lambda = 10\%$

Figures 5.8 and Figure 5.9 show the scalability of AniPQO with the increase of dimension for QT8 and QT9 respectively using log scale. The overhead ratio of FPC and optimization

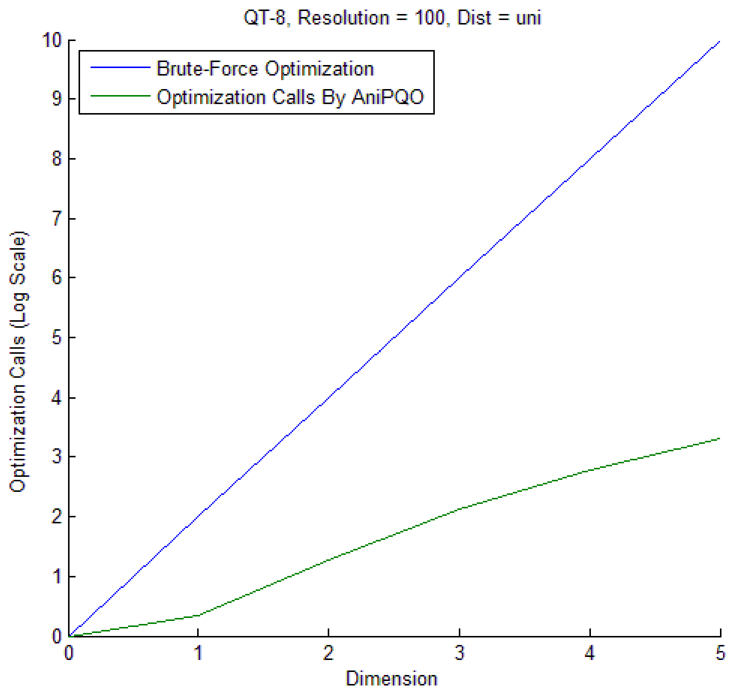


Figure 5.8: Scalability on AniPQO on QT8 with uniform distribution

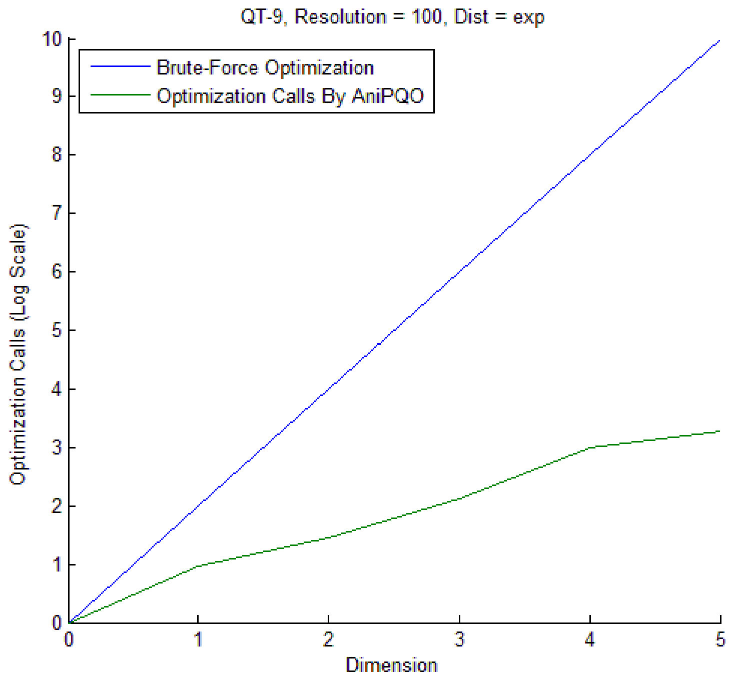


Figure 5.9: Scalability on AniPQO on QT9 with exponential distribution

for commercial optimizers is **1:10** [1]. Hence, we have combined the number of FPC calls in optimization calls to compare efforts made by AniPQO and brute-force approach. Though we have used $\lambda = 10\%$ in case of AniPQO, CG will still need plan diagram, generated by brute-force approach, for the reduction using threshold $\lambda = 10\%$. The graphs in figures show that AniPQO also increases exponentially. But as compared to brute-force approach, AniPQO still saves the time in *an order of magnitude* and scales very well with the increase of dimension.

Chapter 6

Conclusions and Future Work

After evaluating AniPQO using various evaluation scenarios, we have found that with very less number of optimization calls, AniPQO gives Reduced-POSP and this heuristic technique also performs very well (giving significantly less harmed location percentage and max harm) on such industrial-strength optimizer. We conclude that AniPQO can be used to produce on the fly anorexic plan diagrams which can give a rough idea that how will the reduced plan diagram look after the reduction. These plan diagrams can also be used in scenarios where time required to produce reduced plan diagram matters and not bounded cost-degradation. We have evaluated AniPQO upto 3 dimensions, because evaluation will not be practical for higher dimensions. As the future work, AniPQO can be evaluated on other commercial industrial-strength optimizers which support perfect FPC feature and also for higher dimensions by simulating plan-cost-functions.

Bibliography

[1] Harish D. Atreyee Dey, Sourjya Bhaumik and J. Haritsa. Efficiently approximating query optimizer plan diagrams. In VLDB, 2008.

[2] Harish D., P. Darera, and J. Haritsa. On the Production of Anorexic Plan Diagrams. In *VLDB*, 2007.

[3] Goetz Graefe and William J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In ICDE, 1993.

[4] P. Darera Harish D. and J. Haritsa. Identifying robust plans through plan diagram reduction. In VLDB, 2008.

[5] Jayant R. Haritsa. Query optimizer plan diagrams: Production, reduction and applications. In ICDE, 2011.

[6] Arvind Hulgeri and S. Sudarshan. Anipqo: Almost non-intrusive parametric query optimization for nonlinear cost functions. In VLDB, 2003.

[7] Naveen Reddy and Jayant R. Haritsa. Analyzing plan diagrams of database query optimizer. In VLDB, 2005.

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

[9] `www.cse.iitb.ac.in/dbms/Projects/Optimization/`.

[10] `www.tpc.org/tpch`.