AdaNexus: An Improved Nexus Algorithm

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Faculty of Engineering

BY

Achint Chaudhary

Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012 (INDIA)

July, 2020

Declaration of Originality

I, **Achint Chaudhary**, with SR No. **04-04-00-10-42-18-1-15879** hereby declare that the material presented in the thesis titled

AdaNexus: An Improved Nexus Algorithm

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2018-20**.
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 Parents & Close People

For those whom I have lost, wish you can see this

Acknowledgements

I am deeply grateful to Prof. Jayant R. Haritsa for his unmatched guidance. He was and always will be an inspiration to me for his thoughts and his perception of looking at things around us, be it academia or real life. It is my good luck that I got a chance to work with him and obtain few chunks of knowledge from him.

I am thankful to Anshuman Dutt, Srinivas Karthik and Sanket Purandare for their guidance and discussions. I am thankful to my lab mates for providing me with valuable suggestions during this work. My sincere thanks goes to my friends Diksha and Hemant, who have patiently listened and understood many complications and helped me bring out clearer pictures to work with. My heartfelt appreciation goes to CSA and CDS office staff who made the academic functioning smooth with best co-ordination possible.

Finally, I would express my gratitude wholeheartedly to my parents, who have taught me how to stay calm and keep working on what is essential. Nothing I have done or will ever do will match their support that I carry within.

Abstract

Declarative query processing in database systems often leads to sub-optimal performance due to wrong selectivity estimation from those encountered during actual execution. Plan Bouquets is a technique proposed to substitute selectivity estimation by selectivity discovery at run-time, to provide worst case performance guarantees. This is done by performing multiple partial executions for same query in an incremental fashion of cost budget from a bouquet which is compiled at very first stage.

This technique is suitable for OLAP queries as high overheads of optimizing most part of selectivity error space are amortized over multiple invocation of query in OLAP scenarios. Full space exploration overheads in the past are improved upon with NEXUS algorithm, that only discovers points useful for bouquet compilation.

In this work, we proposed an adaptive version of NEXUS named AdaNEXUS, which utilizes geometrical properties of contours to be discovered for bouquet. This algorithm reduces overheads of compilations empirically with the same theoretical worst case performance complexity. Further, we provide upper bounds for maximum cost deviation possible during bouquet compilation due to use of either NEXUS or AdaNEXUS. Evaluation of proposed system is done on TPC-DS benchmark with different scales to test system. It is demonstrated that around an order of magnitude reduction is observed in compilation overheads when compared with NEXUS. Also, quality of contours discovered by AdaNEXUS is better than those discovered by NEXUS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

SQL query processing is declarative in nature, user only specifies what needs to be done, how the intended task will be done is role of underlying system. There are a large number of execution strategies for a SQL query, each called a query plan. All these query plans will yield same results but have high variations in running times. Role of database optimizer is to select optimal (in terms of running cost) query plan. During choice of optimal query plan, database optimizer makes multiple cost based decision to compare different query plans. Cost for each physical operator in a query plan is a function of number of tuple, it processes, known as cardinality. Cardinality normalized in range of $[0, 1]$ is known as selectivity throughout literature. Selectivity estimations of optimizer for identifying the optimal query plan are done using statistical models and meta-data information about schema. Selectivity estimates are often poor due to variety of reasons [14], which results in inflated query response time.

1.1 Background

There are multiple techniques proposed in literature to improve quality of selectivity estimates like better statistical models, on-the-fly re-optimization, etc., but none of them provides bounds on worst case performance guarantees.

An entirely different approach based on run-time selectivity discovery is proposed called *plan bouquets*, which for the first time, provides strong theoretical bounds on worst-case performance as compared to oracular optimal performance possible from all the available plan choices.

For each given query, predicates prone to selectivity error contribute as dimension in *Error-prone Selectivity Space(ESS)*. ESS is a multi-dimensional hyper-cube. The set of optimal plans over the entire range of selectivity values in ESS is called *Parametric Optimal Set of Plans(POSP)*. POSP is generated by asking optimizer's chosen plans at various selectivity locations in ESS using *selectivity injection module.* Cost surface generated over entire ESS is called *Optimal Cost Surface(OCS).* An *Iso-cost Surface(IC)* is a collection of all points from OCS which have same cost of optimal plan at each of these locations cost.

1.2 Motivation

Compilation of plan bouquet is process of drawing iso-cost contours, which involves getting selectivity points and corresponding optimal plans at each point for any contour we are drawing. In experimental setting ESS is discretized at some resolution 'RES' where let 'D' be the dimension of ESS. Number of optimizer calls in entire ESS can reach RES^D. This number is exponential in number of dimensions and results in high overheads of bouquet compilation. NEXUS is an algorithm developed in past to avoid doing optimizer calls on entire ESS, it does so by making optimizer calls only on points lying on contours. Number of optimizer calls made by NEXUS to draw 'm' contours in worst case is twice of $m * RES^{(D-1)}$, which is still exponential in nature.

When 'm' is sufficiently high and 'D' is also high, to keep things computationally feasible a moderate choice of 'RES' is made. When this is the case, contours drawn by NEXUS can suffer from cost deviation from ideal desired cost value which does affect worst case performance guarantees. So it is desired to keep total overhead feasible with acceptable contour cost deviations.

1.3 Contributions

In this work, we have devised algorithms to improve contour discovery for plan bouquet, which constitutes most in compilation overhead. We have contributed in the following two ways

- **Speeding-up contour discovery:** We proposed AdaNEXUS algorithm, which is an improvement over NEXUS, that utilizes geometric properties of iso-cost contours generally observed in practice. This algorithm is designed to reduce total overheads in terms of optimizer calls and also to keep low contour cost deviations then what is there with NEXUS algorithm. Even in absence of geometric properties that AdaNEXUS utilizes to gain speed-up, AdaNEXUS still has same worst case complexity as NEXUS.

- **Contour cost deviation bounds:** We provide upper bounds on contour cost deviation values for both NEXUS and AdaNEXUS, from which it will be clear that AdaNEXUS provides less cost deviation during contour construction than NEXUS and should be preferred over NEXUS.

1.4 Organization

Rest of the thesis is organized as follows:

1. Chapter 2 provides a brief detail on Plan Bouquets technique.
2. Chapter 3 discusses existing bouquet compilation techniques, associated overheads and re-formulation of contour discovery problem.
3. Chapter 4 focuses on design of AdaNEXUS by leveraging geometric properties of OCS and iso-cost contours.
4. Chapter 5 gives worst case cost deviation bounds for both NEXUS and AdaNEXUS.
5. Chapter 6 discusses experimental evaluation of our work.
6. Chapter 7 discusses conclusion of our work and future work.

Chapter 2

Plan Bouquets

Basics of Plan Bouquets from [1] are given in this chapter which is an approach for robust query processing.

2.1 Overview

Plan Bouquet is an approach where compile time selectivity estimation is eschewed by systematic discovery of selectivity values at run-time by multiple partial execution carried in incremental cost-budgeted manner from a subset of POSP called Plan Bouquet.

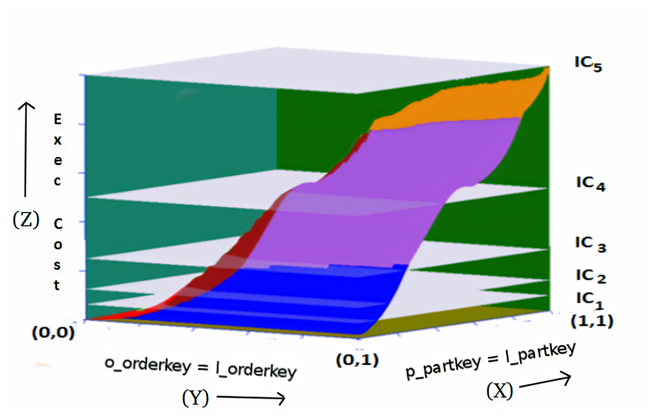


Figure 2.1: OCS and Plan Trajectories intersection

A subset of POSP is identified as *Plan bouquet*, which is obtained by the intersection of plan trajectories with OCS, creating multiple Iso-cost surfaces, each of which is placed at some cost-ratio (r_{pb}) from the previous surface. Following Fig 2.1 [8] depicts an exemplar OCS and its intersection with IC trajetories for a sample 2-Dimensional ESS.

2.2 Worst case performance guarantee of Plan Bouquet

Since each plan on an iso-cost surface has a bounded execution limit, and incurred cost by execution using bouquet will form geometric progression. The figure below shows the performance of 1D plan bouquet w.r.t to optimal oracular performance.

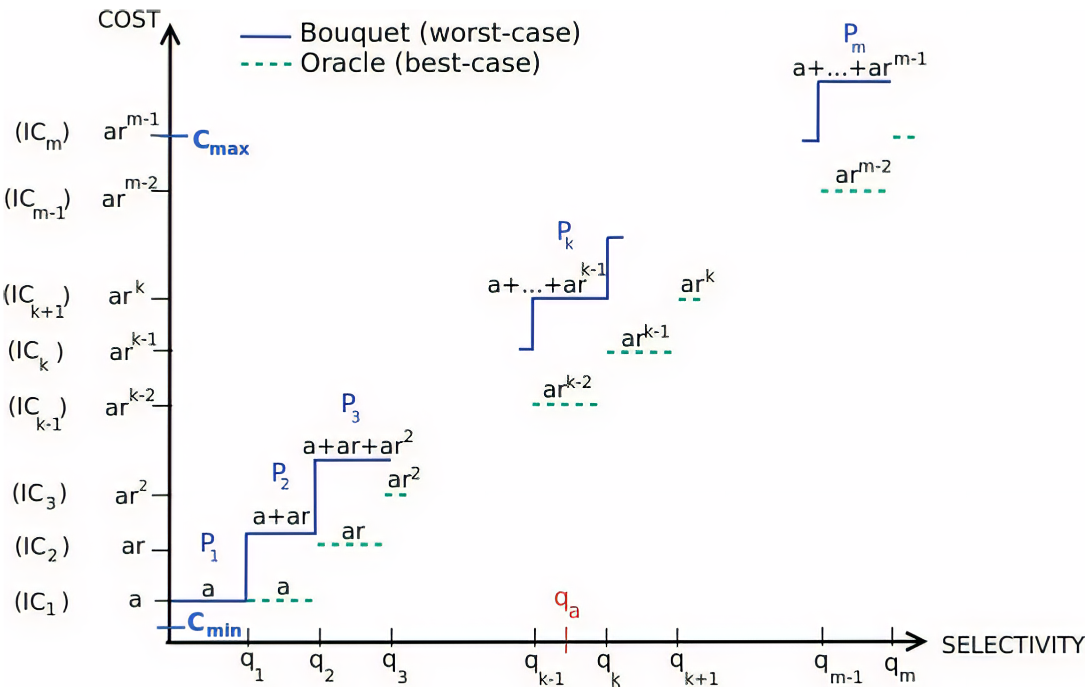


Figure 2.2: Cost incurred (Oracular vs Bouquet)

In the above Fig 2.2 [1], various plans up to actual selectivity value q_a are executed. Each plan has a limit provided by the next iso-cost surface. This yields total execution cost of

$$C_{bouquet} = \sum_{i=1}^{k} cost(IC_i)\ a + a * r_{pb} + a * r_{pb}^2 + ... + +a * r_{pb}^{k-1} = \frac{a*(r_{pb}^k - 1)}{r_{pb} - 1}$$

This leads to sub-optimality (ratio of incurred cost to optimal cost) of plan bouquets approach as follows

$$SubOpt(*, q_a) \leq \frac{\frac{a*(r_{pb}^2-1)}{r_{pb}-1}}{a*r_{pb}^{k-2}} \frac{r_{pb}^2}{r_{pb}-1} - \frac{r_{pb}^{2-k}}{r_{pb}-1} \leq \frac{r_{pb}^2}{r_{pb}-1}$$

This value is minimized using $r_{pb} = 2$, which provides theoretical worst case bound of 4 times the optimal execution time.

Extending the same idea to multiple dimensional ESS, MSO guarantee will become 4ρ, where ρ is maximum cardinality (of plans) on any of iso-cost surface.

Computing value of ρ requires huge compile time effort. Also, it is platform dependent and low value of ρ is desired for practical MSO_g which was obtained using anorexic reduction heuristic at the time plan bouquets was developed.

Later an improved algorithm called *SpillBound*[2], which is able to provide performance guarantee based only on query inspection and is quadratic function in number of error-prone predicates, which is same as dimensionality of ESS. MSO guarantee obtained by SpillBound is

$$D^2 + 3D$$

It is notable that Spillbound provides pre-compilation guarantees independent of ρ, which is platform dependent.

2.3 Impact of Cost Deviation on MSO_g

For any existing or yet proposed contour discovery mechanism, it is not always possible to obtain exact desired cost value CC_i for a contour IC_i that is drawn. Each contour in practice deviates by some value $(1+\alpha)$ from CC_i, which means contour cost will lie in the cost interval $[\ CC_i/(1+\alpha)\ ,\ (1+\alpha)*CC_i\ ]$. We will now see the impact of cost deviation on MSO_g.
Let,
The total number of contours be k (IC_1, IC_k).
Cost of all contours except IC_{k-1} is deviated on higher side by $(1+\alpha)$
Cost incurred due to execution of bouquet sequence is

$$BS_{cost} = CC_{k-1} + (1+\alpha)*(CC_k + \sum_{i=1}^{k-2} CC_i)$$

While optimal cost can be

$$OPT_{cost} = \lim_{h \to 0^+} (CC_{k-1} + h) = CC_{k-1}$$

In this case, MSO_g will be

$$\frac{BS_{cost}}{OPT_{cost}} = \frac{(1+\alpha)*\sum_{i=1}^{k} CC_i - \alpha * CC_{k-1}}{CC_{k-1}}$$

$$= (1+\alpha) * \frac{r_{pb}^2}{r_{pb}-1} - \alpha \leq (1+\alpha) * \frac{r_{pb}^2}{r_{pb}-1} = \eta * \frac{r_{pb}^2}{r_{pb}-1}$$

Using $r_{pb} = 2$, we will get MSO_g

$$4 * (1+\alpha) = 4 * \eta$$

Also, it is worth notable that cost deviation $(1+\alpha)$ or η should always respect

$$1 \leq (1+\alpha) = \eta << r_{pb}$$

Chapter 3

Problem Formulation

3.1 Notations

The following notations are used within the text:

Notation	Description
SP	Selectivity Predicates
EPP	Error Prone Predicates
ESS	EPP Selectivity Space
OCS	Optimal Cost Surface
$POSP$	Parametric Optimal Set of Plans
RES	Resolution of Discretized ESS
Dim or D	Dimensions of ESS
ε_i	Minimum Selectivity of Predicate SP_i
m	Number of Iso-cost Contours
IC_i	i_{th} Iso-cost contour
CC_i	Cost budget of IC_i
r_{pb}	Cost Ratio of Iso-cost contours
$(0,1]$ or $[\varepsilon,1]$	Selectivity interval of Discretized ESS
P_j	Plan with assigned identity j
F_j or PCF_j	Plan Cost Function(PCF) for Plan P_j
$Cost(P,q)$	Cost of plan P at location q in ESS
d_{sel}	Uniform spacing of selectivity in ESS
r_{sel}	Ratio of selectivity values in ESS
β_{max}	Worst case slope of Plan Cost Function
α	Tolerance of contour thickening
γ	Constant for exponential smoothing

Table 3.1: Notations used in text

3.2 Assumptions

3.2.1 Plan Cost Monotonicity(PCM)

This assumption implies that if location q_j spatially dominates location q_i in ESS, cost of optimal plan at location q_j is more than cost of optimal plan at location q_i.

$$(q_j \succ q_i) \Rightarrow (Cost(q_j) > Cost(q_i))$$

This also comes from a simple fact that processing more tuples will incur more cost. We assume that Plan Cost Functions and OCS are continuous and smooth in nature.

3.2.2 Axis Parallel Concavity(APC)

This assumption, as stated in [3], is on Plan Cost Function (F_p) which is not just monotonic but exhibits a weak form of *concavity* in their cost trajectories. For 1D ESS, F_p is said to be concave if for any two selectivity locations q_i, q_j from ESS and any $\theta \in [0, 1]$ following condition holds

$$F_p(\theta * q_i + (1 - \theta) * q_j) \geq \theta * q_i + (1 - \theta) * q_j$$

Generalizing to D dimensions, a PCF F_p is said to be *axis parallel concave* (APL) if the function is concave along every axisparallel 1D segment of ESS. It simply states that each PCF should be concave along every vertical and horizontal line in the ESS. Further, an important and easily provable implication of the PCF exhibiting APC is that the corresponding $Optimal\ Cost\ Surface(OCS)$ which is the infimum of the PCFs, also satisfies APC. Finally, for ease of presentation we will generically use concavity to denote APC in the remainder of this work.

3.2.3 Bounded Cost Growth(BCG)

BCG property as defined by [4], is as follows for plan cost function F_p.

Here, $f(\alpha)$ is an increasing function. Increase in selectivity by $\alpha \geq 1$ will result in maximum cost increase by a factor of $f(\alpha)$. As in the case of APC assumption, BCG is also proven to hold for OCS when it is true for all POSP plan cost functions.

$$F_p(\alpha * q.j) \leq f(\alpha) * F_p(q.j)$$
$$\forall j \in 1, 2, ..., D \wedge \forall \alpha \geq 1$$

They have also claimed that identity function $f(\alpha) = \alpha$ suffices in practice.

3.2.4 Piece-wise Axis Parallel Linearity(APL)

Plan Cost Functions and OCS are shown to be piece-wise linear in [5]. This property commonly comes from the fact that partial derivatives of common physical operators (except the sort operator, which is seldom found in industry strength benchmark [4]) are linear in nature.

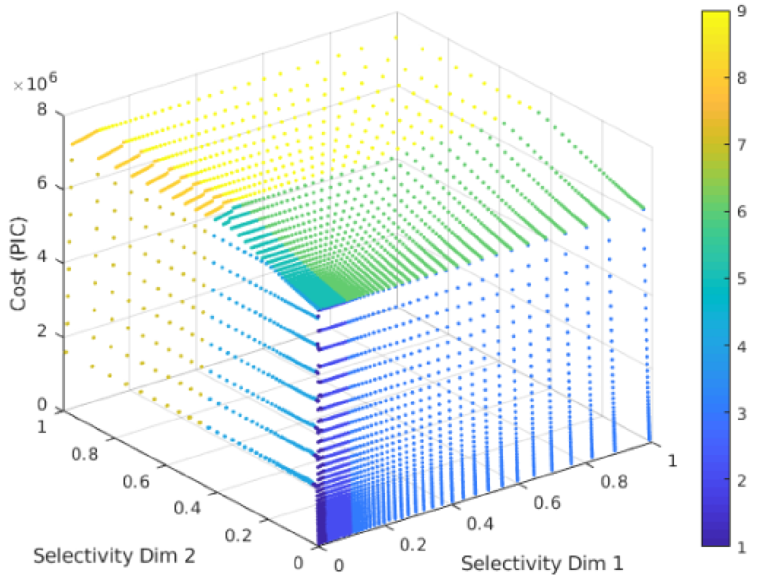


Figure 3.1: Multiple APL functions to fit OCS

When it is the case that OCS or Plan Cost Functions are not truly piece-wise linear, a coarse approximation of piece-wise linear function can still be fitted to them. Similar work has been done in our lab in the past [6].
While in our work, there is no need to fit any such piece-wise linear function. This will reduce our effort of fitting points from entire OCS into a piece-wise APL function which will itself be exponential in nature.

3.2.5 Perfect Cost Model of Optimizer

This assumption states that poor choices of plan come only from the cardinality estimation error of optimizer and not from the cost model itself. While we have assumed perfect cost model of optimizer, an optimizer with bounded cost model will also work well. Improving the cost model is an orthogonal problem. One work on offline tuning [7] proves that the cost model can be tuned to predict value within 30% of the estimated cost values.

3.2.6 Selectivity Independence

We assume that selectivity of predicates is independent of each other. While this is a common assumption in query optimization literature, it often does not hold in practice.

3.3 Existing compilation methods

Next step of compilation is to identify selectivity location and their optimal plans for each of the iso-cost contours. For now, there are two options available for contour construction:

1. Full ESS Enumeration

2. NEXUS (Neighborhood Exploration Using Seed).

3.3.1 Full ESS Enumeration

This is most naïve yet effective approach and will be referred as full space enumeration at most places. In this approach, optimal plan and its cost at all points of ESS is asked from query optimizer.
The points at which cost of optimal plan is equal to cost value of any iso-cost contour is qualified to be added to that contour. This will incur $\mathcal{O}(RES^D)$ optimizer calls. Here, RES is resolution chosen to discretize ESS and Dim is dimension of ESS. Each dimension in ESS represents a error-prone predicate.
This approach is certainly exponential in number of dimensions and a suitable value of RES should be chosen to make overall cost computationally feasible. Full space enumeration can completely exploit parallel architecture of modern multi-core systems available.

3.3.2 NEXUS(Neighborhood EXploration Using Seed)

An optimization over full space enumeration is introduced in plan bouquets [1]. NEXUS is an algorithm proposed to avoid making unnecessary optimizer calls on points lying in between contours. If we have total m iso-cost contours to discover, worst case complexity of NEXUS for entire compilation process can go up to $\mathcal{O}(m * D * RES^{D-1})$.
At first, NEXUS seems to be promising for reducing compilation overhead, but faces following multiple issues [8]:

1. If large number of contours need to be drawn, NEXUS is effectively close to full space enumeration, especially in high dimensional ESS.

2. If a lower bound on selectivity of query predicate is known through domain knowledge, SpillBound can shrink ESS by making this lower bound as origin. However, NEXUS needs to redraw new iso-contours from scratch.

3. Randomized contour placement to introduce fairness in plan bouquets needs more contours to be drawn. This makes NEXUS cumulatively more expensive than full space enumeration.

In worst case, total optimizer calls made by NEXUS is twice the number of points lying on iso-cost contours.
Note: Both of the above methods for finding iso-cost contours make a common assumption that resolution of discretized ESS grid should be sufficiently high such that we can always find contiguous iso-cost locations with cost of optimal plans at these locations lying in interval $[CC_i, (1 + \alpha) * (CC_i)]$ even with small values of α, say, 0.05.
Due to this assumption, we will see some issues which are common to both full ESS enumeration and NEXUS.

3.4 Complexity issues in compilation

Both methods have associated behavior or requirements with them as follows:

1. Complexity is exponential in Dim

2. Need of sufficiently high resolution on each axis

The algorithm with complexity $\mathcal{O}(RES^{Dim})$ becomes computationally unfeasible to run with sufficiently high-resolution dimensions of ESS. To prevent this in practice, instead of going with sufficiently high resolution with uniform distribution of selectivity values on each axis, experiments can be tried to run on low-resolution picture.
Next we will see a potential issue which may arise with the use of low resolution uniformly distributed selectivity values.
But first, we will discuss both methods in brief for a 2D ESS example, to find points lying on contour IC_i with cost CC_i.

1. **Full space enumeration**
   The grid points lying in the interval should be on contour if cost of optimal plan lies within cost interval $[CC_i, (1 + \alpha) * (CC_i)]$.

2. **NEXUS**
   Locate seed $S(x, y)$ and then iterate to find next neighbor until loop ends to find any next

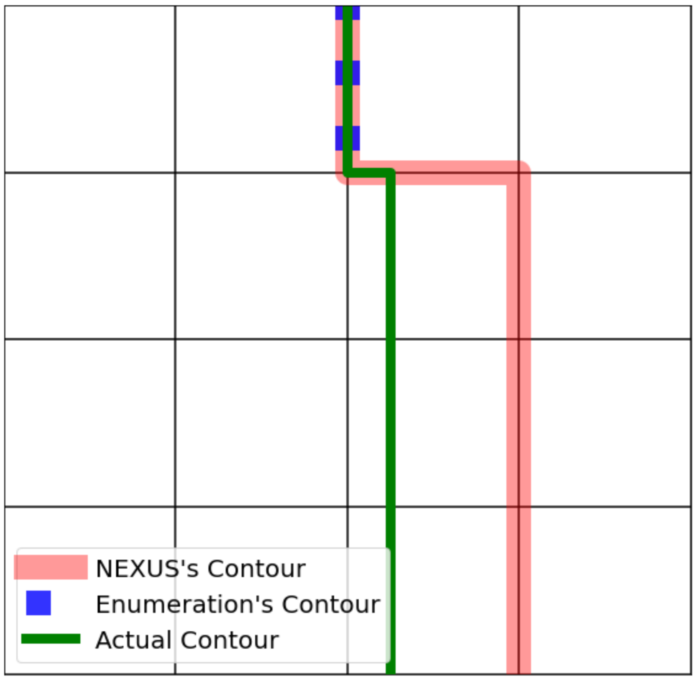
location with given search condition.

```
$while\ (S\ has\ a\ neighbor\ in\ 4^{th}\ quadrant)$:
    $if\ Cost(S(x, y-1)) < CC_i$:
        $S(x, y) = S(x+1, y)$
    $else$:
        $S(x, y) = S(x, y-1)$
```

Algo 1. Neighbor exploration in NEXUS

Due to usage of low resolution and low tolerance factor α, full space enumeration may result into incomplete contour while NEXUS may result into contour with cost inflated more than factor of $(1+\alpha)$. Pictorial representation of potential issues encountered are depicted in Fig 3.2.

Figure 3.2: Contour discovery issue in low resolution ESS



To avoid this yet keeping computational feasibility, one possible option is to raise the value of α but that does impact MSO_e. Also, from observation and from APC, we know most changes in slope will happen close to origin.

So, to (empirically) avoid above explained issue, when working with low resolution and high dimensional ESS and to keep cost computationally feasible, geometric distribution of selectivity value was used on each axis of ESS in practice.
This use of low resolution and geometric distribution is never explicitly stated in literature and may violate MSO_e in practice. This violation is not observed yet, but proof for the same is also pending like a *conjecture*. Rationale for using geometric distribution in selectivity space till now is that it captures many points in low selectivity values and most changes in plan choices take place in low selectivity values.
For making a geometric distribution to work, there are numerous hyper-parameters to tune. The methods, tips and techniques along with their impact on contours discovered are the missing part from literature which we will try to provide and prove in a systematic way.

3.5 Problem Statement

Given a D-dimensional ESS, construct all contours for plan bouquet in cost budget ratio of r_{pb} in a time efficient manner, reducing both number of optimizer calls and cost deviation from desired cost for each contour without empirically affecting the performance guarantee of plan bouquet technique.

Chapter 4

Leveraging Contour Geometry

4.1 Where NEXUS can be improved

NEXUS in worst case makes number of optimizer calls twice the number of points lying on contour in high resolution discretized ESS. So idea of NEXUS is to eliminate unnecessary optimizer calls on points lying between contours, but this optimization fails and cost close to full-space enumeration when we have moderate resolution to work with and large number of contours needs to be discovered.
From past works [5][10], contours are observed to be either piece-wise linear or approximated to be piece wise linear. See Fig 4.1 for reference of contours generated on a 50GB TPC-DS with Query instance Q91.

4.2 AdaNEXUS design

We have attempted to utilize this highly piece-wise linear nature of contours to get improved and faster version of NEXUS, namely AdaNEXUS, which should in practice speed up the contour discovery process, the main cost overhead of compilation.
We have removed discretization of ESS at very first place. AdaNEXUS considers ESS as continuous hyper-cube but a minimum fixed step size is used as a substitute for discretization that still provides more flexibility than discretization as first step.
So, initial seed discovery using binary search of NEXUS is replaced with piece-wise linear interpolation search. This lead to better quality seed discovery as starting point for AdaNEXUS.
Now we will look at design of contour exploration of AdaNEXUS.
As an example, consider the red contour (which is 4^{th} contour in the diagram). The seed, as usual, will be located on top boundary of ESS.
Now, what if we can magically get the slope of contour in ESS space (do not consider cost into

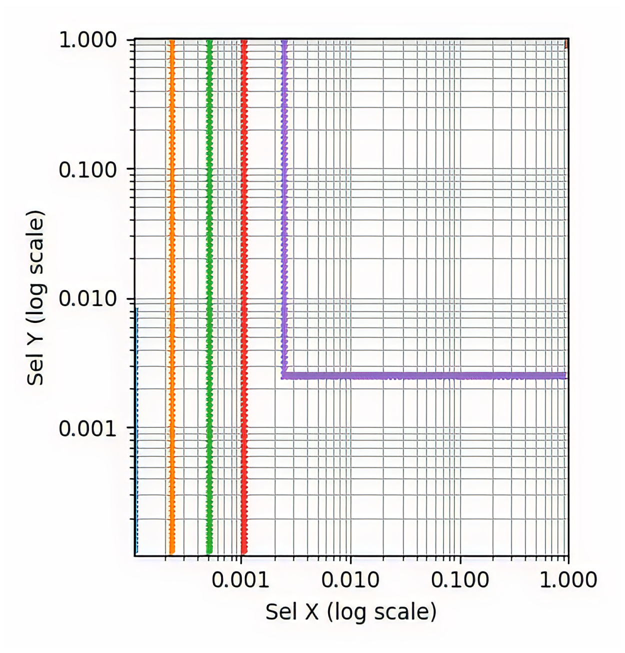


Figure 4.1: Q91 contours on 50GB TPC-DS

picture), which is nothing but infinity, as contour is parallel to Y-axis.

We could have used *exponential search* to reduce number of points to be discovered on the contour where optimizer calls are made. In the best-case complexity will change from $\mathcal{O}(RES)$ to $\mathcal{O}(\log(RES))$.

There are some fundamental issues with this approach:

1. Slope in ESS space for any piece of piece-wise linear contour is not known a priori.
2. Even if exact slope can be approximated, exponential search may miss some plans on contours due to large steps taken.

First, we will see how to overcome the second issue in our idea.

We will be using a bisection exploration to find if we can find a different plan in between two successive points discovered by exponential search. If different plans are obtained in either half, a recursive function is called until either an interval on bisection exploration has the same plans on both endpoints or interval length vanishes. Same idea is formulated in form of pseudo-code as follows:

```
def $bisectionEXP(left, right)$ :
    $P_L, P_R = P_{opt}(left), P_{opt}(right)$
    $if\ (left \leq right)\ and\ (P_L\ != P_R)$ :
        $mid = \frac{left+right}{2}$
        $P_M = P_{opt}(mid)$
        $if\ (P_L != P_M)$ :
            $bisectionEXP(left,\ mid)$
        $if\ (P_R != P_M)$ :
            $bisectionEXP(mid,\ right)$
```

Algo 2. Bisection exploration to find plans missed by exponential search

Now, let's come to the first issue of our search approach, which is getting slope of each piece of piece-wise linear function. Let us pose it as an online control system problem with feedback and fallback strategies.

As we know even with original NEXUS and full ESS exploration, a tolerance interval of $[CC_i, (1+\alpha) * CC_i]$ (with sufficient low value of α, say, 0.05) is used and points are chosen such that surface thickening is avoided, i.e., points must be chosen as close as possible to lower bound of search cost interval.We will exploit similar idea to search within cost interval $[CC_i/(1+\alpha), CC_i * (1+\alpha)]$ and our search method will try to pick the point having optimal cost lying in mid of specified cost interval.

NEXUS solves D-dimensional contour discovery by recursively solving 2-dimensional sub-problems, and this is shared with AdaNEXUS also.

With the knowledge of seed, we will start from one end of contour and will explore neighborhood location in ESS. Slope information will be obtained on-the-fly and tuned based on deviation from mid-value of cost interval so that search will always lie within the given interval. When search goes beyond the tolerable cost interval, we always have a fallback to last valid point and try with half the step size taken in last wrong decision.

Sometimes, even at the minimum possible step-size, the present tuned direction vector can make the search go outside the cost interval. In this case, an aggressive search for the next direction will be done in first, third and fourth quadrants without any tuning. This sets a new tuned direction vector to continue search further.

This will not be an exact exponential search procedure but is expected to run much faster than linear step sizes in earlier NEXUS which exploits very less information about geometry of contour. This method of dynamic tuning of slope with exponential steps and finding points missed in between using bisection exploration will require lesser optimizer calls for piece-wise linear contours, which is expected to be observed in practice.

Next we have given a pseudo-code for AdaNEXUS in Algo 3. *S, C, P,* Δ correspond to selectivity, cost, plan and direction vector to continue search respectively. *now* in this context refers to current value, e.g. P_{now} refers to optimal plan at current point on contour, *next* refers to next point we are discovering on contour.

Cost() function returns both cost and optimal plan at some selectivity location in ESS if P_{opt} is given in argument. *Correct()* function is used to find new direction to search based on correction done with information from proxy location. *TuneDir()* function does devised exponential smoothing, *bisectionEXP()* does bisection exploration when plans at two end points are different. At the end *Rotate()* is a function to find direction when tuned direction vector (Δ_{now}) is unable to find a point on contour even with smallest possible size.
Note: All corrections and additions done in AdaNEXUS are in logarithmic selectivity space, and will result in multiplicative change in selectivity values. Same is done when NEXUS is applied in ESS discretized with exponential distribution.

```
def $AdaNEXUS(CC_i)$ :
    $S_{now}, P_{now} = InitializeSeed(CC_i)$
    $C_{now} = Cost(P_{opt}, S_{now})$
    $step, \Delta_{now} = 1, [0, -1]$
    $while\ (There\ exist\ next\ point)$ :
        $S_{proxy} = S_{now} + step * \Delta_{now}$
        $C_{proxy}, P_{proxy} = Cost(P_{opt}, S_{proxy})$
        $S_{next}, \Delta_{next} = Correct(CC_i, C_{proxy})$
        $C_{next}, P_{next} = Cost(P_{opt}, S_{next})$
        $if\ (\max\left(\frac{C_{next}}{CC_i}, \frac{CC_i}{C_{next}}\right) \leq (1+\alpha))$ :
            $\Delta_{now} = TuneDir(\Delta_{now}, \Delta_{next}, step)$
            $bisectionEXP(S_{now}, S_{next})$
            $S_{now}, step = S_{next}, 2 * step$
        $else$ :
            $if\ step > 1$ :
                $step = step / 2$
            $else$ :
                $\Delta_{now} = Rotate(S_{now}, CC_i)$
```

Algo 3. AdaNEXUS with Bisection exploration, direction tuning and angle correction

We have not mentioned a strict condition on searched points so that they lie exactly in between the cost interval $[CC_i/(1+\alpha), CC_i * (1+\alpha)]$ to avoid surface thickening like NEXUS.

$Q - Error$ [10] is chosen as error function to make AdaNEXUS algorithm discover points having cost close to mid value of tolerable cost interval. An online tuning algorithm based on *PID Control* [11] is used for tuning direction vector which is referred as *step-size adaptive exponential smoothing.*

4.3 Step-size adaptive exponential smoothing

Like integral term in *PID control* is used to get a smooth search instead of a zig-zag path. Tuning the direction vector during contour discovery in AdaNEXUS also requires us to use multiple errors made during process contour discovery.

A simple functioning like that of integral term is achieved using exponential averaging. Δ is used to denote the direction vector. Simple exponential average is formulated as

Let, Δ_{now} and Δ_{avg} be immediate Δ and average Δ

$$\Delta_{avg} = \gamma * \Delta_{now} + (1-\gamma) * \Delta_{now}$$

$$for\ some,\ 0 < \gamma < 1$$

Since we are taking steps of variable sizes during exponentiation, a simple exponential average assigns same impact value to each point, which should not be the case.
So considering the present step size is **S**, the impact of Δ_{now} on tuned direction vector can be better formulated as

$$\gamma * \Delta_{now} + \gamma * (1-\delta) * \Delta_{now} + ... + \gamma * (1-\delta)^{(S-1)} * \Delta_{now}$$

$$= (\gamma * \tfrac{(1-(1-\gamma)^S)}{(1-(1-\gamma))}) * \Delta_{now} = (1-(1-\gamma)^S) * \Delta_{now}$$

This leads us to modified expression for step-size adaptive exponential smoothing, which is used in tuning Δ_{avg} in AdaNEXUS search and given as

$$\Delta_{avg} = (1-(1-\gamma)^S) * \Delta_{now} + ((1-\gamma)^S) * \Delta_{avg}$$

Chapter 5

Cost Deviation Bounds

This chapter provides upper bounds on cost deviation bounds due to NEXUS and AdaNEXUS

We have seen at end of chapter 2 that contour cost deviation has a linearly proportional impact on MSO_g. Also, we have designed the conceptual framework for AdaNEXUS. In this chapter we will prove the theoretical upper bounds on contour cost deviations of NEXUS and AdaNEXUS, and will try to prove that if in theory AdaNEXUS provides lesser cost deviation during contour construction than NEXUS.

5.1 Cost Deviation with NEXUS

Assume $\beta_{max} = 1$ in BCG assumption, r_{sel} is the ratio at which ESS is discretized, r_{cost} be cost ratio change per step in ESS

$$r_{cost} \leq \beta_{max} * r_{sel} \rightarrow r_{cost} \leq \beta_{max} * r_{sel}$$

Also discretization of ESS in interval $[\varepsilon, 1]$ is done at a ratio

$$r_{sel} = (\sqrt[(RES-1)]{\varepsilon})^{-1}$$

Consider Fig 5.1 where NEXUS is discovering contour of cost CC_i. Let all black points have cost C_k due to inherent deviation due to discretization, where

$$C_k = lim_{h \rightarrow 0^+}(\frac{CC_i * r_{sel}}{1 + h})$$

Then, the yellow point will have cost less than CC_i. In that case, blind search of NEXUS will pick a next red point.

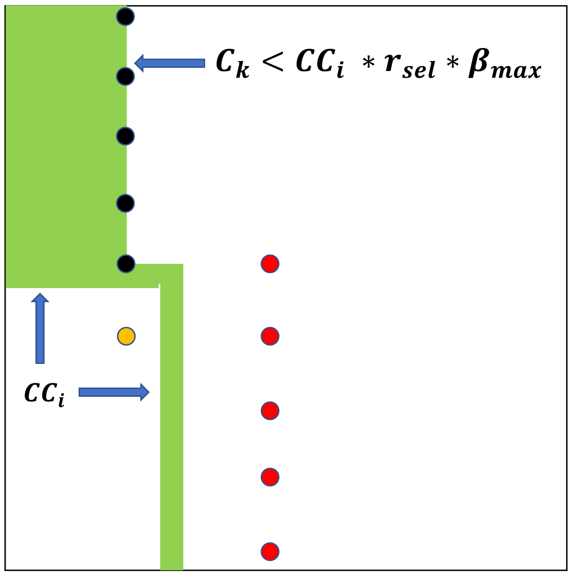


Figure 5.1: Worst case scenario for NEXUS cost deviation

Cost of red point be C_r, where C_r due to BCG is

$$C_r = C_k * r_{sel} = lim_{h \to 0^+}(\frac{CC_i * r_{sel}}{1+h}) * r_{sel} \le CC_i * r_{sel}^2 = CC_i * \eta$$

So, we can claim that worst case cost deviation for NEXUS is

$$\eta = (1 + \alpha) = r_{sel}^2$$

Or in a general case when β_{max} is not known but yet BCG assumption holds

$$\eta = (1 + \alpha) = (\beta_{max} * r_{sel})^2$$

5.2 Cost Deviation with AdaNEXUS

AdaNEXUS does not apply explicit discretization but does limit itself to a minimum step size to be taken in ESS. These steps are multiplicative in nature, and during 2-dimensional exploration AdaNEXUS will take a step in either of two predicates or partial steps to change value of both selectivities at once, keeping the constrain that L1-norm of step size is still of unit measure.

Let, β_i and β_j be partial derivatives of OCS w.r.t selectivity predicates sp_i and sp_j respectively.
Then cost change in one step will be,

$$(\beta_i * r_{sel})^{(w)} * (\beta_j * r_{sel})^{(1-w)}$$

For any,

$$w \in [0, 1]$$

Since both β_i and β_j are upper bounded by β_{max}

$$(\beta_i * r_{sel})^{(w)} * (\beta_j * r_{sel})^{(1-w)} \leq \beta_{max} * r_{sel}$$

So, for AdaNEXUS maximum cost deviation is

$$\eta = (1 + \alpha) = (\beta_{max} * r_{sel})$$

It can be noted that NEXUS has larger cost deviation specially for higher value of r_{sel} than AdaNEXUS.

Chapter 6

Experimental Evaluation

Now we will move towards profiling the performance of our proposed methods over existing ones in literature. Overheads incurred during bouquet compilation are discussed in terms of optimizer calls, while quality of contours discovered is measured in terms of cost deviation from ideal contour cost.

Database environment: Queries for evaluation are taken from TPC-DS benchmark that covers wide spectrum of join geometries including *star, chain,* etc. Number of base relations varies from 3 to 6. Number of error prone predicates varies from 3 to 5, all of which are join selectivity errors. Since physical schema has indexes on all columns, which lead to more variety of plans possible, making it difficult to achieve robustness due to large ratio of C_{max}/C_{min}. TPC-DS benchmark is used at different scale, varying from 1GB to 250GB. Benchmark metadata is scaled (keeping the same distribution) for each scale using CODD[13].

System environment: Database engine used in our experiments is modified version of Postgres-9.4. Hardware platform is vanilla HP-Z4-G4 workstation with Intel Core i9-7900X CPU, 32GB DDR4 2666MHz RAM and 2TB disk storage. Now we will compare different frameworks of bouquet compilation and also our devised algorithms over existing algorithms like NEXUS.

6.1 BCG Validation

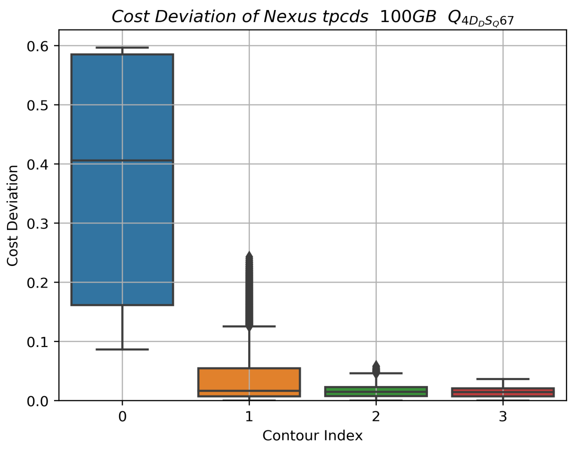
We have verified that bounded cost growth does hold in all our queries on different scales of database instances we have experimented upon. To find out β_{max}, we did optimizer calls in close neighborhood of all end points of ESS, this took total cost of

$$\Theta(2^{(2*Dim)})$$

We have observed that claim of [4] does hold, and β_{max} is bounded by 1.0 in all our experiments. This is due to the fact that contribution from Sort is very less compared to the total cost of a plan in industrial strength benchmarks[4]. This bound on β_{max} is then used to calculate cost deviation bound during contour discovery.

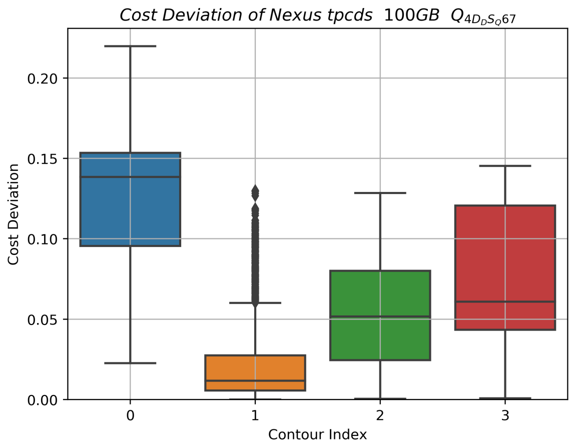
6.2 Uniform vs Exponential Distribution

We have evaluated both uniform distribution and exponential distribution of selectivity values to discretize ESS before applying NEXUS with a moderate resolution to make compilation time feasible (few hours for 5 dimensional queries), cost deviation values (α) for each contour for a Q67 using uniform and exponential distribution is given in Fig 6.1. These are later referred as AP NEXUS and GP NEXUS respectively, since selectivity values on each axis of ESS are in arithmetic and geometric progression respectively.



[h!]

(a) Uniform distribution



(b) Exponential distribution

Figure 6.1: Comparison of Cost deviation for Q67 compilation

It is notable that $(1+\alpha) \leq r_{pb}$ and uniform distribution will lead to high value of α. For some queries in our test suite, we have observed that this assertion of $(1+\alpha) \leq r_{pb}$ is not respected. Hence we can conclude that uniformly spaced selectivity values should not be used even for discretization of ESS with moderately high resolution.

6.3 Tuning exponential smoothing

We have given an expression for step-size adaptive exponential smoothing at end of chapter 4 where γ is constrained in interval (0, 1). We have empirically observed that γ=0.7 gives best results during our experiments. In the further experiments we have used the same value as smoothing constant. Fig 6.2. gives an idea about different values of smoothing constant and present step-size on contribution of latest direction vector towards tuned vector.

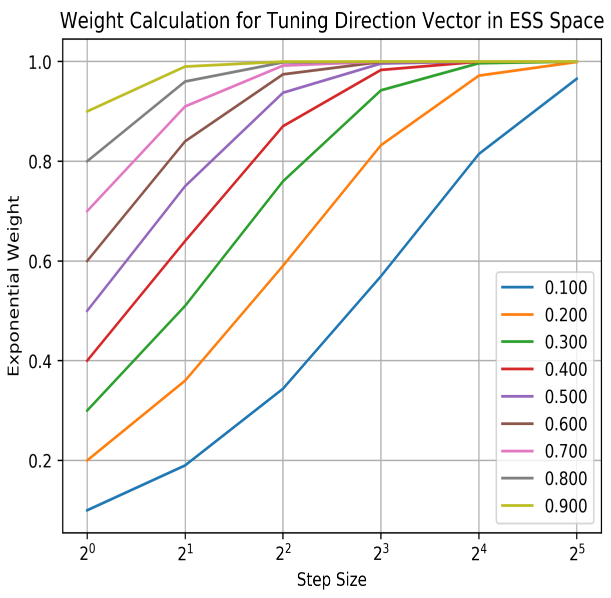
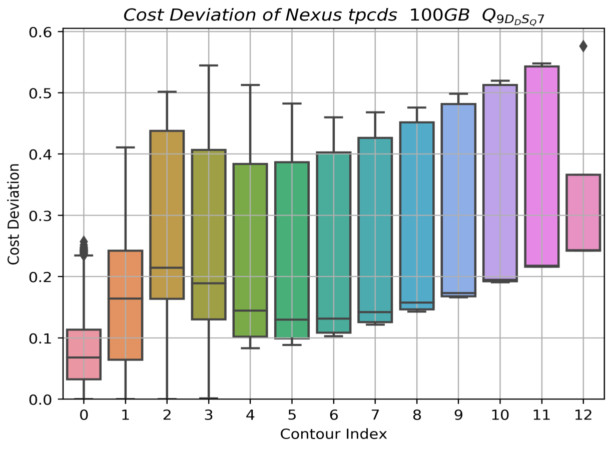


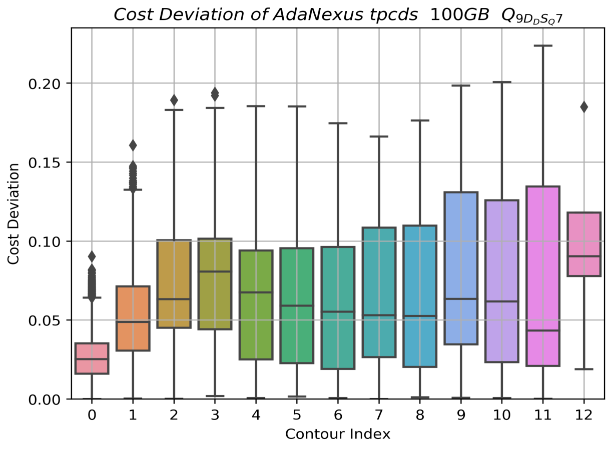
Figure 6.2: Step-size and γ impact on tuning

6.4 Deviation bounds due to Exponential Distribution

In the end of chapter 5, we have come up with cost deviation bounds when using either exponential discretization with NEXUS or exponential step size into continuous ESS space for AdaNEXUS with an example of Q7 which is a 4-dimensional query. Below Fig 6.3. shows contour-wise cost deviation distribution for NEXUS and AdaNEXUS respectively.



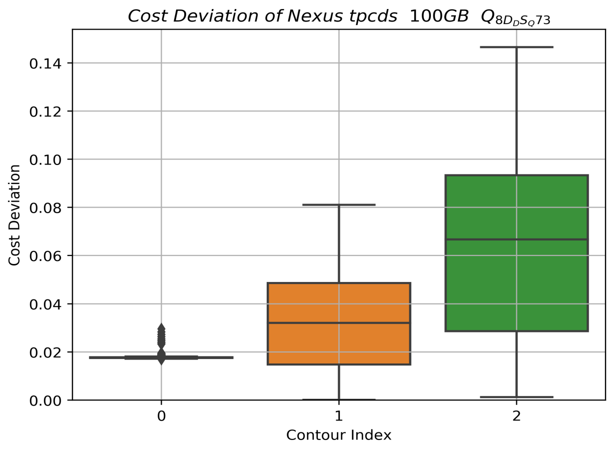
(a) NEXUS with exponentially discretized ESS



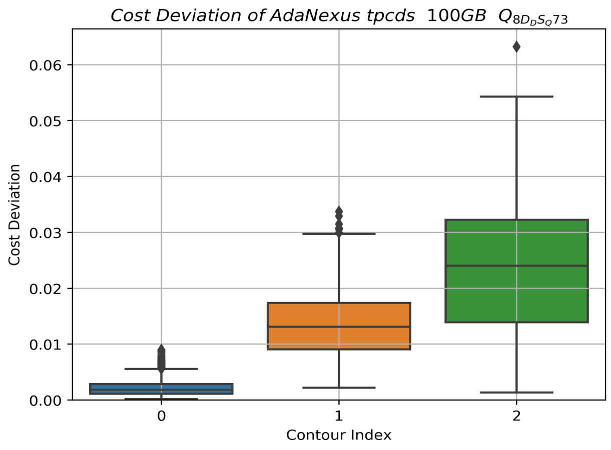
(b) AdaNEXUS with exponential steps in ESS

Figure 6.3: Cost deviation for NEXUS vs AdaNEXUS on Q7 TPC-DS 100GB

This behavior of cost deviation reduction is not just observed in queries where NEXUS has high cost deviation, but also on queries having low deviation with NEXUS. As an example we will see Q73.



(a) NEXUS with exponentially discretized ESS



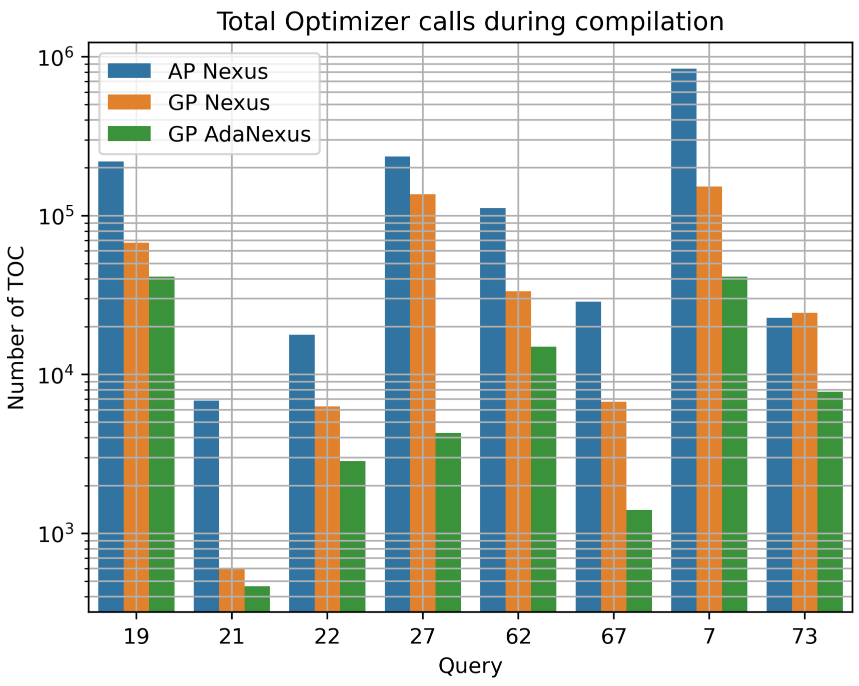
(b) AdaNEXUS with exponential steps in ESS

Figure 6.4: Cost deviation for NEXUS vs AdaNEXUS on Q73 TPC-DS 100GB

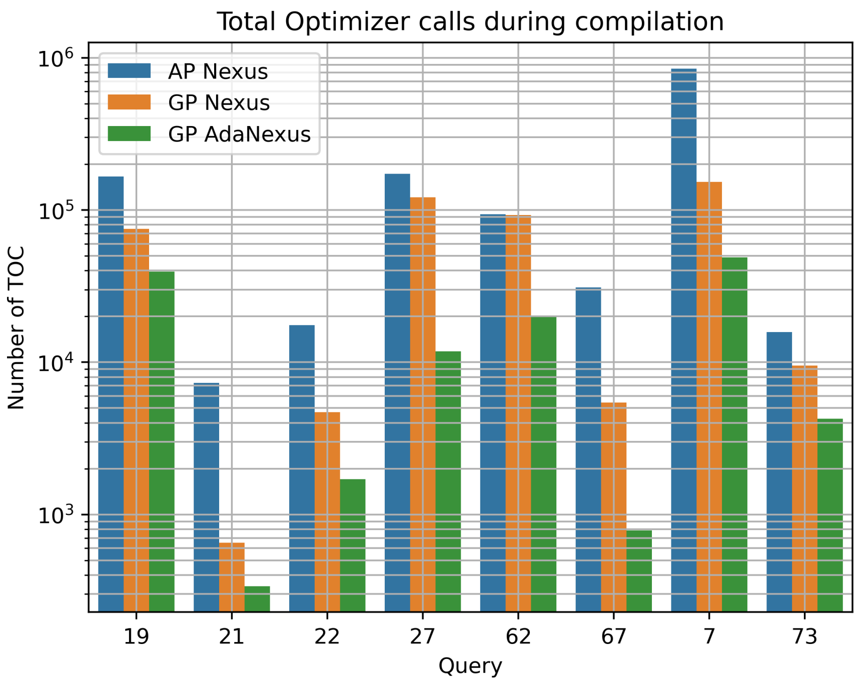
We have observed in all our experiments that cost deviation bounds and empirically observed deviation are lesser for AdaNEXUS as compared to those obtained using NEXUS. These deviations in contour cost have a directly proportional impact on MSO_g similar to λ in anorexic reduction [9] (however that provides a gain by reducing plan cardinality on each contour).

6.5 Overhead reduction using AdaNEXUS

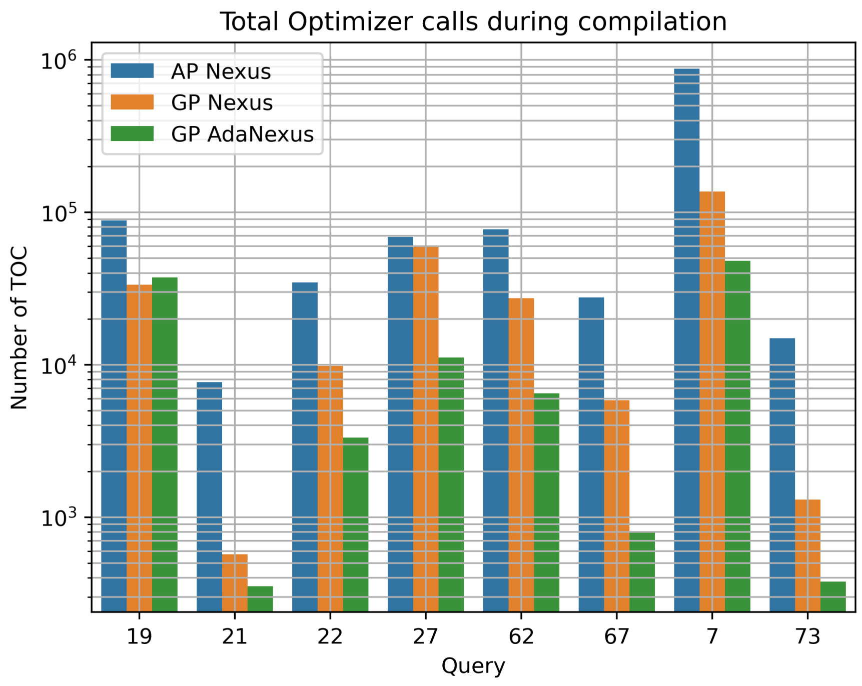
The main role of AdaNEXUS is to reduce overhead in compilation process, where reduced cost deviation is just a plus point of AdaNEXUS design. Now we will see a comparison of overheads incurred using different bouquet compilation approaches.



(a) TPC-DS 1GB



(b) TPC-DS 10GB



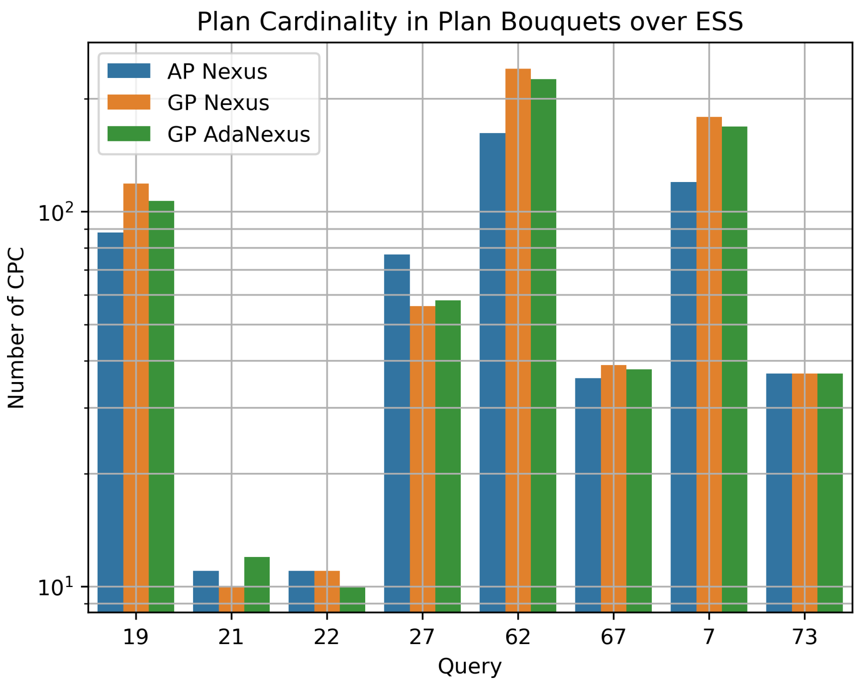
(c) TPC-DS 100GB

Figure 6.5: Optimizer calls using different compilation approaches

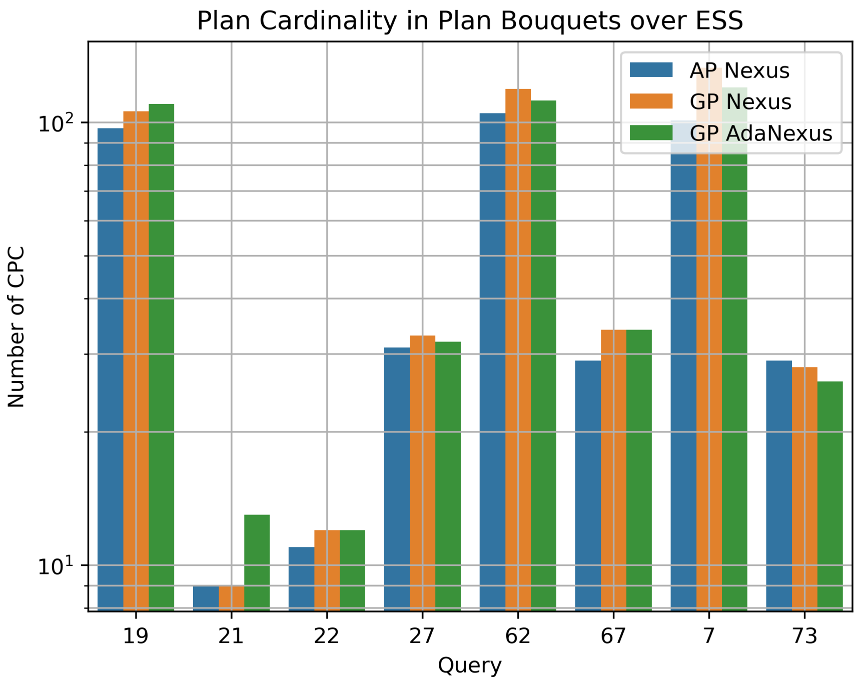
AdaNEXUS brings approximately an order of magnitude reduction in compilation efforts even with moderate minimum step sizes. Also as a bonus, it reduces deviation observed in contour discovery.
An important note here is that *AP NEXUS*, the execution of NEXUS with uniformly spaced selectivity values, results in higher cost deviations that makes it useless to provide useful MSO_g for plan bouquets, specially in higher dimensions.

6.6 Plan Cardinalities

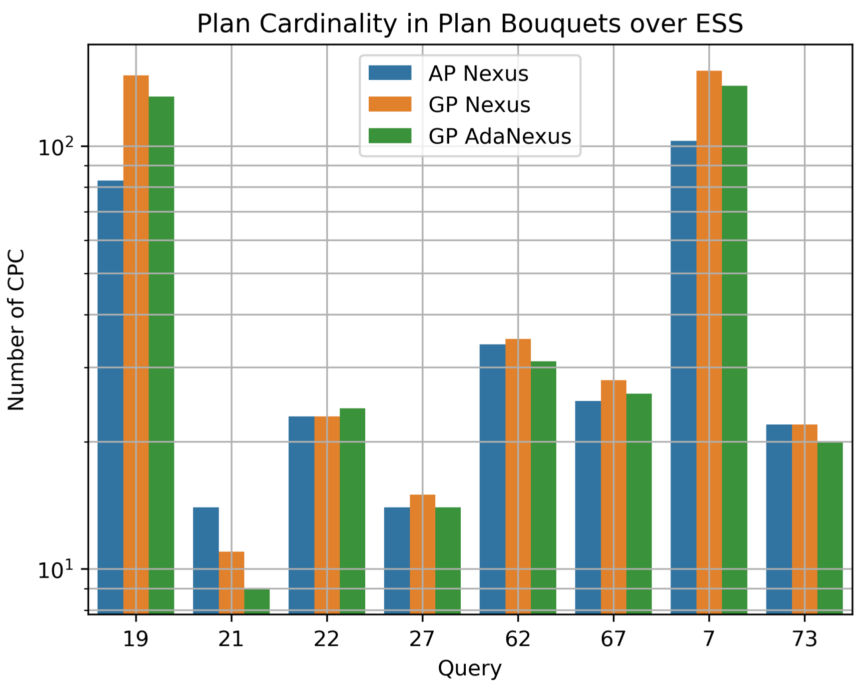
It is not claimed that AdaNEXUS along with binary exploration will find out all plans on contours, that were otherwise observed using NEXUS.
So, we have observed number of plans for all queries and reported the same using both uniform and exponential distribution in NEXUS and AdaNEXUS.



(a) TPC-DS 1GB



(b) TPC-DS 10GB



(c) TPC-DS 100GB

Figure 6.6: Total contour plan cardinalities

Note that we have calculated each plan on different contour multiple times, so above picture can be better understood in terms of number of executions on all contours. In general AdaNEXUS finds out almost all executions discoverable by NEXUS with geometric progression.

So, an empirical claim can be made that almost all plans on each contour discoverable by NEXUS can be discovered also with AdaNEXUS.

Additionally now either a weighted greedy algorithm for anorexic reduction can be deployed before plan bouquet execution, or a contour density independent algorithm like *SpillBound* can utilize AdaNEXUS as a better substitute to NEXUS.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

We have proved (with an additional assumption of BCG) that exponential distribution in ESS discretization leads to usable cost deviation in contour discovery. We have given bounds on cost deviations and validated that they are followed in all our experiments.
Next, we have devised AdaNEXUS with complete removal of ESS discretization which leads to contour discovery with lesser cost deviations and also with lesser optimizer calls.

7.2 Future Work

Both NEXUS and AdaNEXUS use 2-dimensional seed exploration as a subroutine. This way solution to multi dimensional contour discovery is obtained by merging the results of multiple low dimensional sub-problems in a recursive manner. Solutions to all these sub-problems do not share any information with each other e.g. related to slope of adjacent sub-problems being solved. So, AdaNEXUS can gain further speed-up using information shared by related sub-problems already solved. Hence, contour discovery can gain speed with a dynamic programming style solution instead of a simple divide and conquer based search.
Tuning during contour discovery can be improved using a full PID control with parameter tuning using machine learning techniques (as parameter tuning is crucial in PID, even when they are suitable for optimizing linear processes). Also, some full-fledged machine learning algorithms (preferably robust and interpretable) can be used to speed up contour discovery process.
Given proper framework to work on discretization of ESS with exponential distribution, incremental algorithms can be devised in conjunction with AdaNEXUS, as additional work in database scale-up is expected to be sub-linear in scale-up.

Bibliography

[1] Anshuman Dutt and Jayant R. Haritsa. *Plan bouquets: A fragrant approach to robust query processing.* In ACM Trans. on Database Systems (TODS), 41(2), pages 1–37, 2016. 4, 5, 11

[2] Srinivas Karthik, Jayant R. Haritsa, Sreyash Kenkre and Vinayaka D. Pandit. *Platform-independent Robust Query Processing.* In Proc. of the 32nd Intl. Conf. on Data Engg., ICDE '16, pages 325-336, 2016. 6

[3] Srinivas Karthik, Jayant R. Haritsa, Sreyash Kenkre and Vinayaka D. Pandit. *A Concave Path to Low-overhead Robust Query Processing.* In Proc. of the VLDB Endow., 11(13), pages 2183-2195, 2018. 9

[4] Anshuman Dutt, Vivek Narasayya, and Surajit Chaudhuri. *Leveraging re-costing for online optimization of parameterized queries with guarantees.* In Proc. of the 2017 ACM SIGMOD Intl. Conf., pages 1539–1554, 2017. 9, 10, 24, 25

[5] Arvind Hulgeri and S. Sudarshan. *Parametric query optimization for linear and piecewise linear cost functions.* In Proc. of the 28th Intl. Conf. on Very Large Data Bases, VLDB '02, pages 167–178, 2002 10, 15

[6] Sanket Purandare, Srinivas Karthik and Jayant R. Haritsa. *Dimensionality Reduction Techniques for Robust Query Processing.* Technical Report TR-2018-02, DSL CDS/CSA, IISc, 2018. DSL Website 10

[7] Wu, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Hakan Hacigumus, and Jeffrey F. Naughton. *Predicting query execution time: Are optimizer cost models really unusable?.* In Proc. of the 29th IEEE Intl. Conf. on Data Engg., ICDE '13, pages 1081– 1092, 2013. 10

[8] Srinivas Karthik V. *Geometric Search Techniques for Provably Robust Query Processing.* PhD thesis, Indian Institute of Science Bangalore, December 2019. 4, 11

[9] D. Harish, Pooja N. Darera, and Jayant R. Haritsa. *On the production of anorexic plan diagrams.* In Proc. of the 33rd Intl. Conf. on Very Large Data Bases (VLDB'07). 1081–1092. 28

[10] Guido Moerkotte, Thomas Neumann, Gabriele Steidl. *Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors.* In Proc. of the VLDB Endow., 11(13), pages 2183-2195, 2009. 15, 19

[11] Wikipedia contributors. (2020, March 12). *PID controller.* In Wikipedia, The Free Encyclopedia. Retrieved 09:25, April 11, 2020. 19

[12] Anshuman Dutt. *Plan bouquets: An Exploratory Approach to Robust Query Processing.* PhD thesis, Indian Institute of Science Bangalore, August 2016..

[13] Ashoke S. and J. Haritsa. *CODD: A Dataless Approach to Big Data Testing* PVLDB Journal, 8(12), August 2015. 24

[14] M. Stillger, G. Lohman, V. Markl and M. Kandil, ”LEO - DB2's LEarning Optimizer”, VLDB, 2001. 1