Robust Contour Discovery for Plan bouquets

Achint Chaudhary

June 30, 2020

MTech Project Report

Abstract

OLAP applications often require a certain set of canned queries to be fired on database with varying the constants in query templates. For optimal execution of these queries, query optimizer does select a strategy known as the query execution plan. These choices are based on cardinality estimates of various predicates that often hugely differ from actual cardinality values encountered during execution. Due to this reason, optimizer choice leads to high inflation in actual execution cost as compared to predicted cost during optimization.

An altogether different approach for query processing was proposed in 2014, named Plan bouquets [1], which is based on selectivity discovery at run-time by repeated cost bounded execution of carefully chosen set of plans. This technique provides strong bounds independent of data distribution. Plan bouquets is suitable for canned queries as compilation overhead for bouquet is amortized over multiple executions.

In this work we have proposed a mathematical framework for qualitative analysis of bouquet compilation, and also proposed algorithm named *AdaNEXUS* improving both qualitative and quantitative aspects of compilation process.

1 INTRODUCTION

Database query optimizer chooses a plan comprising various structural choices of logical and physical operators for query execution. These choices are based on the cost of each operator which is calculated using number of tuples, it processes, known as *cardinality*. Cardinality normalized in range of [0, 1] is known as *selectivity* throughout literature.

These selectivity values are estimated before query execution based on some statistical models used in classical cost-based optimizers. 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.

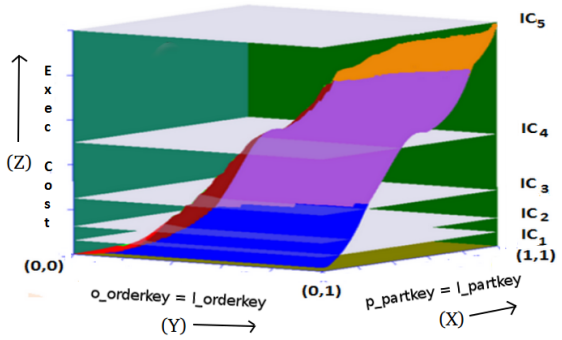


Fig 1. OCS and Plan Trajectories intersection

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 1.[8] depicts an exemplar OCS

and its intersection with IC trajectories for a sample 2-Dimensional ESS.

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.

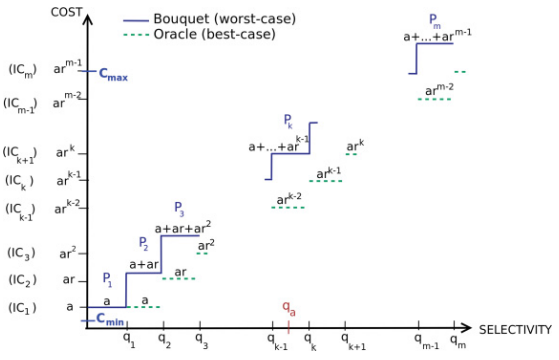


Fig 2. Cost incurred (Oracular vs Bouquet)

In the above Fig 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

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

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 PROBLEM FORMULATION

2.1 Notations

Notation	Description
SP	Selectivity Predicates
WKP	Well Known 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 $[\epsilon, 1]$	Selectivity interval of Discretized ESS
P_j	Plan with assigned identity j
F_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

2.2 Assumptions

2.2.1 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.

2.2.2 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.

2.2.3 Bounded Cost Growth(BCG)

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

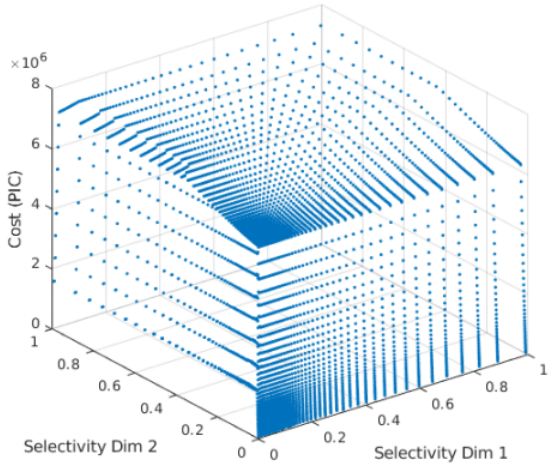
Here, $f(\alpha)$ is an increasing function. Increase in selectivity by $\alpha \geq 1$ will result in maximum cost increase by a factor of $f(\alpha)$. As in the case of APC assumption, BCG is also proven to hold for OCS when it is true for all POSP plan cost functions.

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

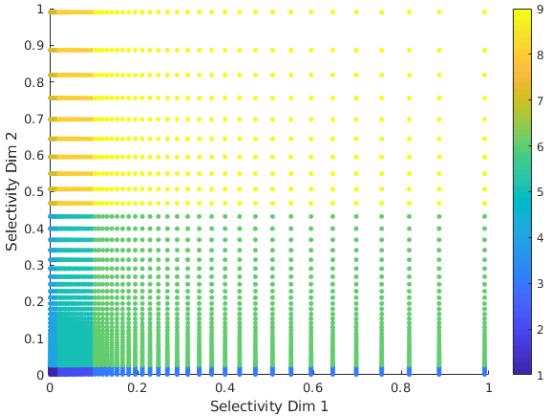
They have also claimed that identity function $f(\alpha) = \alpha$ suffices in practice.

2.2.4 Piece-wise Axis Parallel Linearity(APL)

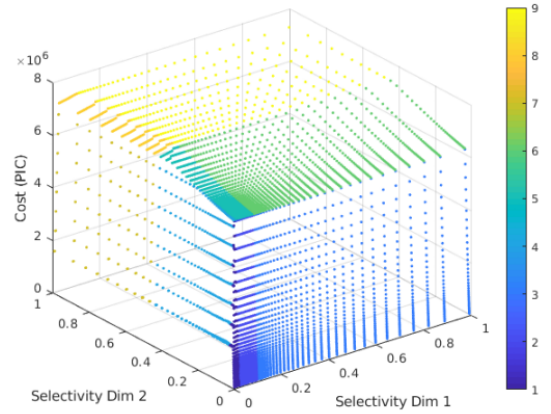
Plan Cost Functions and OCS are shown to be piece-wise linear in [5]. This property commonly comes from the fact that partial derivatives of common physical operators (except the sort operator, which is seldom found in industry strength benchmark [4]) are linear in nature.



(a) Original OCS



(b) Partitioned OCS Domain



(c) OCS fitted with Piecewise functions

Fig 3. 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.

2.2.5 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.

2.2.6 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.

2.3 Performance Metrics

2.3.1 Sub-Optimality

It is the ratio of cost incurred due to wrong selectivity estimation, as compared to the optimal cost possible when actual selectivity is known a priori:

$$SubOpt(q_e, q_a) = \frac{Cost(P_{opt}(q_e), q_a)}{Cost(P_{opt}(q_a), q_a)}$$

Here, both $q_e, q_a \in ESS$.
This definition can be extended to plan bouquets where multiple executions take place in a sequence BS with their respective budgets. So, the definition will be:

$$SubOpt(*, q_a) = \frac{\sum_{exec \in BS} Budget(exec)}{Cost(P_{opt}(q_a), q_a)}$$

2.3.2 Worst case Sub-Optimality

Worst case sub-optimality is sub-optimality w.r.t to q_e that causes maximum suboptimality over entire ESS. This is devised for classic optimizer-based model as follows:

$$SubOpt_{worst}(q_a) = \max_{q_e \in ESS} SubOpt(q_e, q_a)$$

2.3.3 Maximum Sub-Optimality

Global worst case is defined with all possible combinations of q_e and q_a over ESS that results in maximum sub-optimality which is formulated as follows:

$$MSO = \max_{q_a \in ESS} SubOpt_{worst}(q_a)$$

MSO for a sequence of execution from bouquets will be

$$MSO = \max_{q_a \in ESS} SubOpt(*, q_a)$$

Theoretical guarantee is denoted as MSO_g and empirical MSO obtained is denoted as MSO_e.

2.4 Maths of Compilation

Origin and *Terminus* are the extreme ends of ESS, with having minimum possible and maximum possible selectivity of each error-prone predicate respectively. The optimal cost at both these points obtained via selectivity injection is denoted as C_{min} and C_{max} respectively. The number of iso-cost contours (m) using these two values is obtained as follows:

$$m = \lceil \log_{r_{pb}}(\frac{C_{max}}{C_{min}}) \rceil + 1$$

These m iso-cost contours are drawn at r_{pb} cost ratio successively from C_{min} which is referenced as the first term of cost geometric progression (a). Cost value of last contour IC_m may be at lesser cost ratio than r_{pb} from cost budget of IC_{m-1}.
Here is a proof by cases that this will not impact MSO_g.
Case 1. Actual selectivity is discoverable up to execution of plan from IC_{m-1} or any contour before than that, let that contour be IC_i. In that case for 1D plan bouquet, sub-optimality is:

$$\begin{aligned} SubOpt(*, q_a) &= \frac{a*r_{pb}^0 + a*r_{pb}^1 + ... + a*r_{pb}^{i-2} + a*r_{pb}^{i-1}}{a*r_{pb}^{i-2}} \\ &\leq \frac{\frac{a*(r_{pb}^i - 1)}{r_{pb} - 1}}{a*r_{pb}^{i-2}} = \frac{r_{pb}^2}{r_{pb} - 1} - \frac{r_{pb}^{2-i}}{r_{pb} - 1} \\ &\leq \frac{r_{pb}^2}{r_{pb} - 1} \end{aligned}$$

Case 2. Actual selectivity is discovered on execution of plan from IC_m. In that case, let this ratio of IC_{m-1} to IC_m be r_{last}. Also $r_{last} \leq r_{pb}$. Now using expression for sum of geometric progression, MSO_g for plan bouquets is evaluated as follows:

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

$$= r_{last} + \frac{r_{pb}}{r_{pb}-1} - \frac{r_{pb}^{2-m}}{r_{pb}-1}$$
$$\leq r_{last} + \frac{r_{pb}}{r_{pb}-1}$$

To maximize this upper bound, we substitute upper bound of r_{last} with r_{pb}. Hence, resulting expression will be

$$r_{last} + \frac{r_{pb}}{r_{pb}-1} \leq r_{pb} + \frac{r_{pb}}{r_{pb}-1} = \frac{r_{pb}^2}{r_{pb}-1}$$

When $r_{pb} = 2$ is substituted in the final expression, maximum sub-optimality in that case is also upper bounded by 4(which is bound for case 1, bound of plan bouquets in general). This same proof can be easily extended for multi-dimensional plan bouquets with upper bound of $4 * \rho$, where ρ is maximum number of plans on any contour.
Hence, we can state that if the last contour is placed at a cost less than r_{pb} it will provide the same MSO_g.

2.5 Compilation Methods and Overheads

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).

2.5.1 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.

2.5.2 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.

2.6 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*):

$$\textit{if } Cost\big(S(x, y-1)\big) < CC_i:$$

$$S(x, y) = S(x+1, y)$$

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 4.

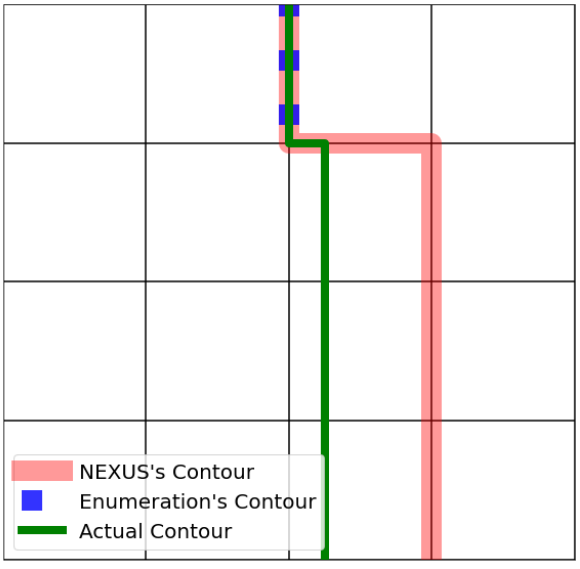


Fig 4. Contour discovery issue in low resolution ESS

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 CONTRIBUTIONS

3.1 Increasing compilation efficiency

Few comments as discussed in [8] highlight some issues with usage of NEXUS. Two significant issues from those are:

1. Same effective cost as full ESS enumeration
2. Need to redraw contours from scratch if lower bound on any selectivity predicate is known

Fraction of speed-up of NEXUS over full space enumeration is

$$\mathcal{O}(\tfrac{RES}{m*Dim})$$

As number of dimensions (Dim) and number of contours to draw (m) increases, both are cases having queries with a greater number of EPP. Also, to make things feasible we choose low values of RES. All this aspect of moving towards tractable compilation time brings NEXUS close to full space enumeration.˙ We will propose an upgrade in NEXUS to make it much faster than its competitor naive algorithm.
But before we start working on a improved version of NEXUS, we will first try to look at second argument made against NEXUS.

3.1.1 Impact of known lower bound

Consider a 1-dimensional example of plan bouquet. If lower bound on predicate selectivity known a priori is δ , then selectivity interval will reduce from $(0, 1]$ to $[\delta, 1]$.
Let selectivity locations for each of contour IC_i be q_i. So,

$$IC = \{IC_1, IC_2, IC_3, ...IC_m\}$$
$$Q = \{q_1, q_2, q_3, ...q_m\}$$

Here, $q_1 = 0.0$ and $q_m = 1.0$
Let δ lie beyond q_k, and actual selectivity q_a be discovered upon execution of IC_i.
So, contours IC_1 through IC_k are no longer needed. Expression for sub-optimality will be changed from

$$SubOpt(*, q_a)$$

$$= \frac{ar_{pb}^{i-1} + ar_{pb}^{i-2} + \cdots + ar_{pb}^{1} + ar_{pb}^{0}}{ar_{pb}^{i-2}}$$

to

$$SubOpt(*, q_a)$$

$$= \frac{ar_{pb}^{i-1} + ar_{pb}^{i-2} + \cdots + ar_{pb}^{1} + ar_{pb}^{k-1}}{ar_{pb}^{i-2}}$$

$$= \frac{\left(r_{pb}^{i} - r^{k-1}{}_{pb}\right)}{(r_{pb} - 1) r_{pb}^{i-2}}$$

$$\leq \frac{r_{pb}^{2}}{(r_{pb}^{2} - 1)}$$

This is the same expression for MSO_g as seen in earlier contours. So, we can now state that with knowledge of lower bound of a predicate's selectivity, we can discard contours with points having selectivity of that predicate less than known lower bound and still achieve same MSO_g.
Hence, now when we are clear that knowledge of lower bound neither impacts NEXUS nor full space enumeration, contours drawn in the past can be continued to use without any change due to knowledge of lower bound.
We will revisit NEXUS and see scope of improvement based on some geometric properties and try to improve existing NEXUS algorithm. Base idea of NEXUS is first locating a seed, which is one end point of contour; use this seed to discover adjacent points of contour lying in 4^{th} quadrant in its neighborhood. Below example borrowed from [1] shows working of NEXUS in pictorial way in Fig 5.

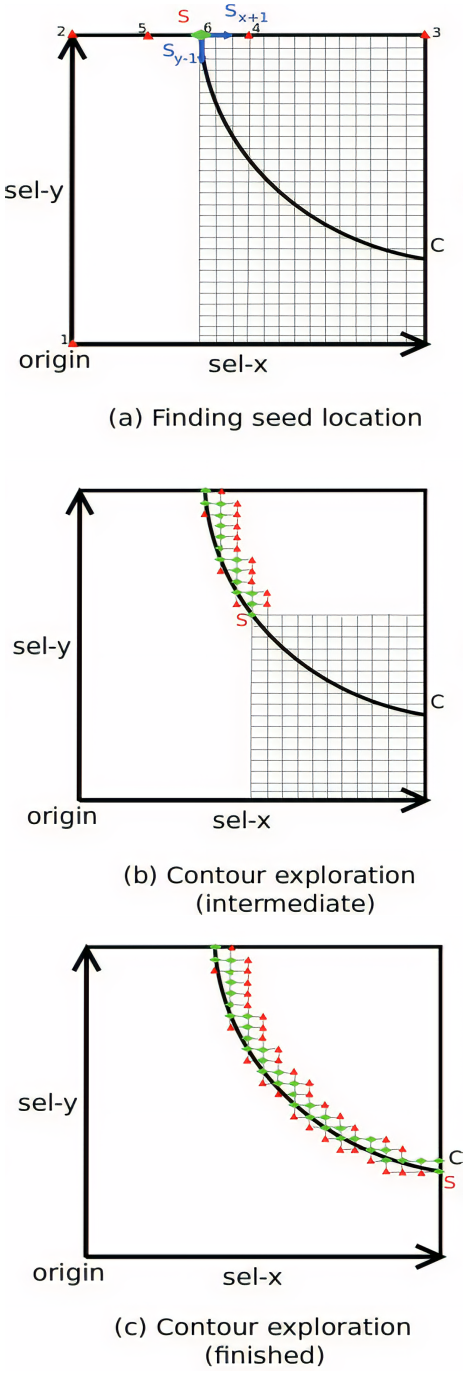


Fig 5. Working of NEXUS

NEXUS in worst case makes number of optimizer calls twice the number of points lying on contour in high resolution discretized ESS.

This at first glance looks somehow smart, but we can still improve it using piece-wise linear geometry of contours.
From past works [5][10], contours are observed to be either piece-wise linear or approximated to be piece-wise linear. See figure for reference of contours generated on a 50GB TPC-DS with Query instance Q91.

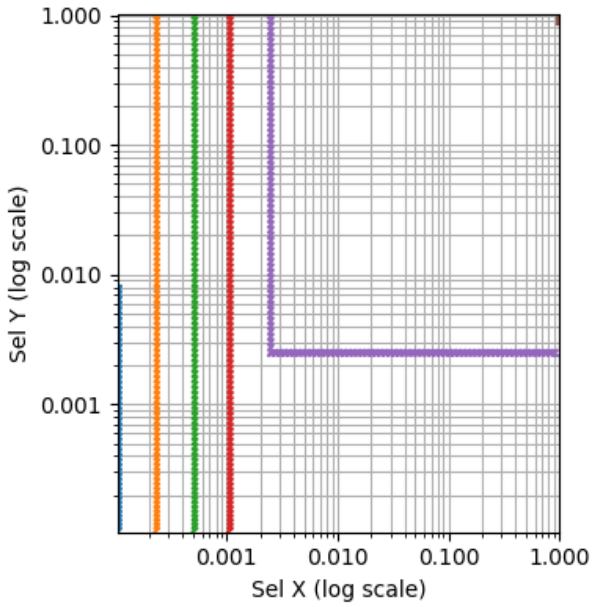


Fig 6. Q91 contours on 50GB TPC-DS

3.2 AdaNEXUS

We have attempted to utilize this highly piecewise 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 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 search 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 search 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 ≤ right) and (P_L != P_R):
        mid = (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 search 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, Δ_now = 1 , [−1, 0]
    while (There exist next point) :
        S_proxy = S_now + step * Δ_now
        C_proxy, P_proxy = Cost(P_opt , S_proxy)
        S_next , Δ_next = Correct(CC_i, C_proxy)
        C_next, P_next = Cost(P_opt , S_next)
        if (max(C_next / CC_i , CC_i / C_next) ≤ (1 + α)) :
            Δ_now = TuneDir(Δ_now , Δ_next , step)
            bisectionEXP(S_now , S_next)
            S_now , step = S_next , 2 * step
        else :
            if step > 1 :
                step = step / 2
            else :
                Δ_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.*

3.3 Geometric progression to discretize each axis of ESS with bounded MSO_g

Earlier full space exploration and NEXUS needs uniform selectivity distribution with high resolution, in practice. But from past work on concavity [3], we know that most of changes in cost value happen close to origin.
To capture this behavior of OCS, we can go with following two options:

1. Sufficiently high resolution with uniform distribution on each axis.
2. Selectivity values should be chosen carefully in geometric progression with bounded relaxation in contour cost.

For high dimensional queries, it is not possible to go with first choice of high resolution. In this sub-section, we will work on usage of selectivity values on each axis as a geometric distribution and obtain relaxed MSO bounds using geometric distribution to discretize each axis of ESS.
Use of Geometric progression with bounded slope of OCS for any predicate SP_i will result in relaxed but bounded contour cost deviation. Let us prove the same.
Formula for finding the last term of GP is

$$b = a * r^{n-1}$$

Let ε be the minimum selectivity to start with on axis corresponding to predicate SP.

$$1 = \varepsilon * r_{sel}^{(RES-1)}$$

From slope bounded cost growth, we know

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

Ratio of selectivity values on axis of ESS is

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

NEXUS is a blind search algorithm with a basic fallback to increase selectivity value of one of predicate during 2-dimensional sub-ESS exploration. With fixed discretization of ESS a priori used in NEXUS, blind search will lead to maximum cost deviation during countour discovery as follows

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

While AdaNEXUS does not rely on fallback strategy this way, cost deviation in case of AdaNEXUS will be

$$\eta = (Dim - 1) * (\beta_{max} * r_{sel})^1$$

It is shown in BCG literature that $\beta_{max} = 1$ generally suffices in practice.

3.4 Reduced FPC calls for Anorexic Reduction

Under the AdaNEXUS framework when it is expected to get lesser number of points (optimizer calls) on some contour, number of points given to each plan is not a good representative for anorexic reduction to take place.
But original anorexic reduction algorithm (which is a simple greedy algorithm) can be modified by assigning weights to each point equal to the distance between it and last point during contour discovery.
Below we provide a simple proof for number of FPC calls made during anorexic reduction and show that, it is directly proportional to number of points discovered on contour.

Given,
Plans in space $= \{P_1, P_2, P_3, \dots, P_n\}$
Total points in space be $S_{Total} = S_1 + S_2 + S_3 + \cdots S_N$

 # FPC Calls for Plan P_1
$S_2 + S_3 + \cdots + S_N = S_{Total} - S_1$

So,

$$Total\ \#\ FPC\ Calls = \sum_{k=1}^{n} (S_{Total} - S_k)$$

$$= n * S_{Total} + \sum_{k=1}^{n} S_k = (n-1) * S_{Total}$$

However it is important to note that *Spillbound*, which was an algorithm invented later, does not require low contour density and can work without anorexic reduction.

3.5 Step-size adaptive exponential smoothing

Like integral terms 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

$$Let, \Delta_{now} \text{ and } \Delta_{avg} \text{ be immediate } \Delta \text{ , and average } \Delta$$

$$\Delta_{avg} = \alpha * \Delta_{now} + (1 - \alpha) * \Delta_{avg}$$
$$for\ some, 0 < \alpha < 1$$

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

$$\alpha * \Delta_{now} + \alpha * (1 - \alpha) * \Delta_{now} + \cdots + \alpha * (1 - \alpha)^{S-1} * \Delta_{now}$$
$$= \left(\alpha * \frac{(1 - (1 - \alpha)^S)}{1 - (1 - \alpha)}\right) * \Delta_{\text{now}} = (1 - (1 - \alpha)^S) * \Delta_{\text{now}}$$

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

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

Note: α used in this subsection refers to parameter used for exponential smoothing and is different from cost deviation parameter used throughout.

4 Experimental Validation

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.

4.1 BCG Validation

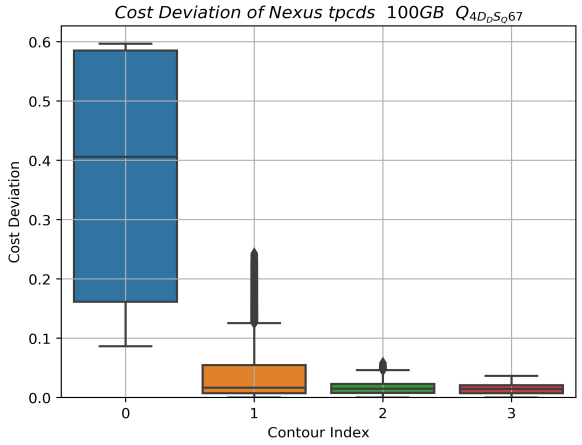
We have verified that bounded cost growth does hold in all our queries on different scales of database instances we have experimented upon. To find out β_{max}, we did optimizer calls in close neighborhood of all end points of ESS, this took total cost of

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

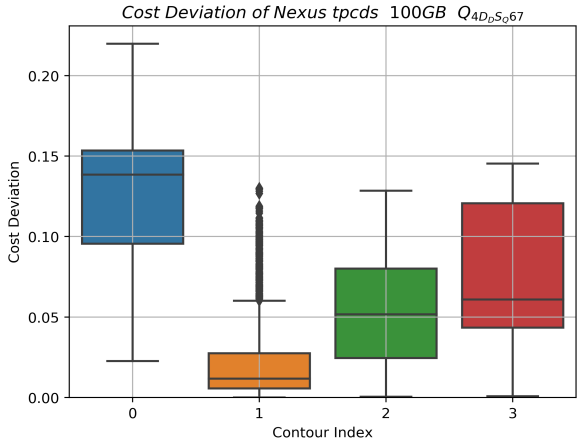
We have observed that claim of [4] does hold, and β_{max} is bounded by 1.0 in all our experiments. This is due to the fact that contribution from Sort is very less compared to the total cost of a plan in industrial strength benchmarks[4]. This bound on β_{max} is then used to calculate cost deviation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

4.2 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 7.



(a) Uniform distribution



(b) Exponential distribution

Fig 7. 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.

4.3 Tuning exponential smoothing

We have given an expression for step-size adaptive exponential smoothing at end of section 3.5 where α is constrained in interval (0, 1). We have empirically observed that α=0.7 gives best results during our experiments. In the further experiments we have used the same value as smoothing constant. Fig 8. gives an idea about different values of smoothing constant and present step-size on contribution of latest direction vector towards tuned vector.

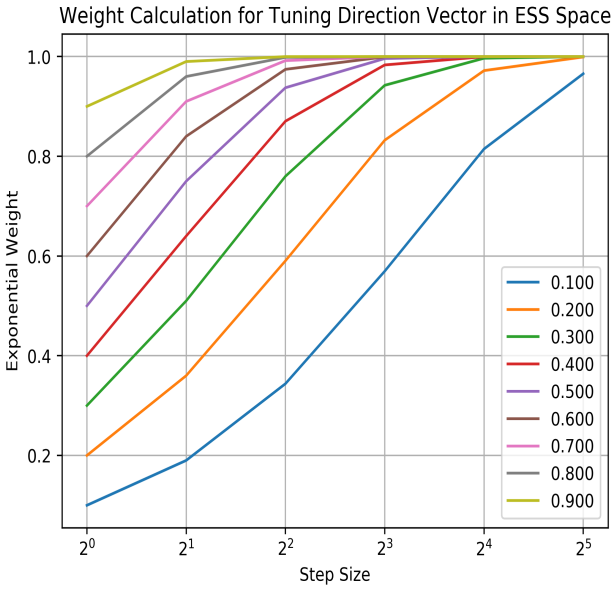
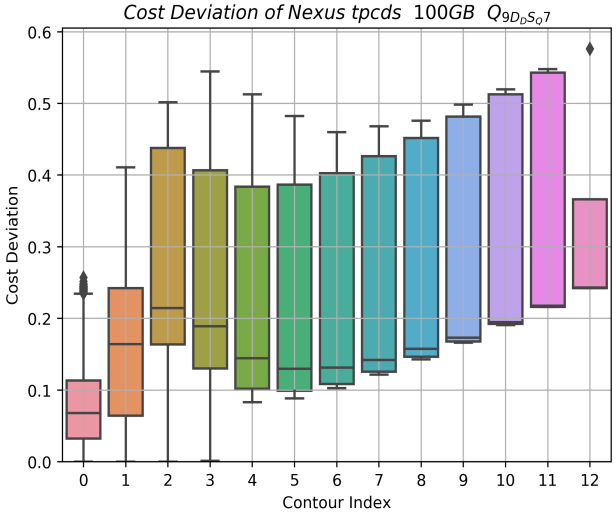


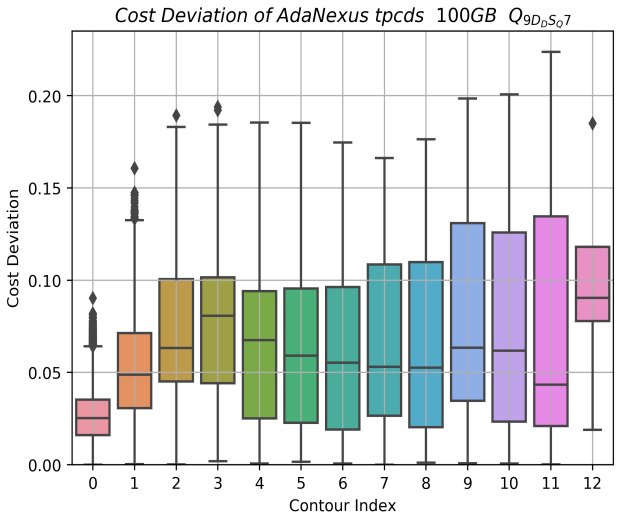
Fig 8. Step-size and α impact on tuning

4.4 Deviation bounds due to Exponential Distribution

In the end of section 3.3, 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 9. shows contour-wise cost deviation distribution for NEXUS and AdaNEXUS respectively.



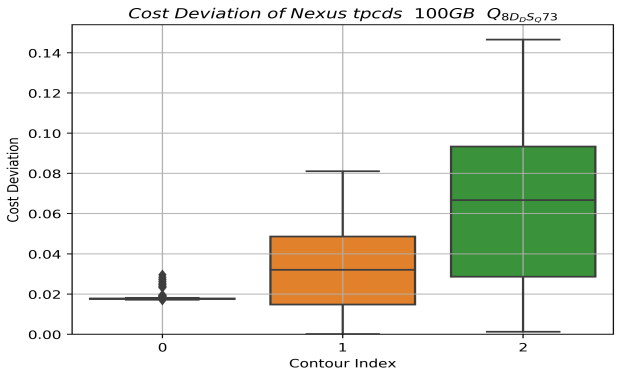
(a) NEXUS with exponentially discretized ESS



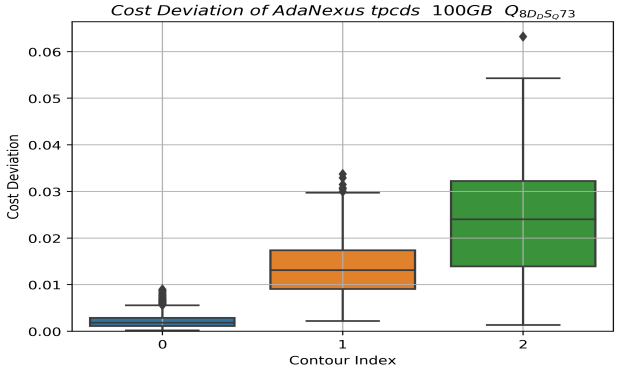
(b) AdaNEXUS with exponential steps in ESS

Fig 9. Cost deviation for NEXUS vs AdaNEXUS

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

Fig 10. Cost deviation for NEXUS vs AdaNEXUS

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_e.

4.5 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.

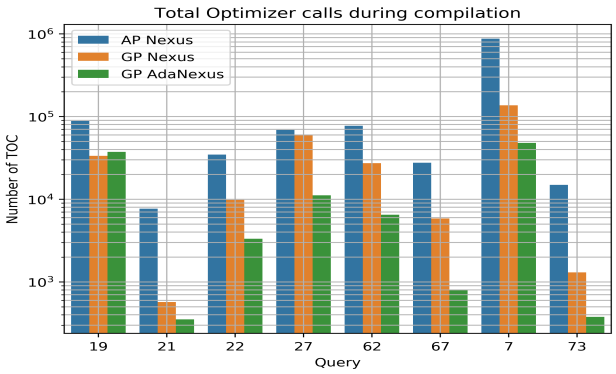


Fig 11. Overheads using different compilation approaches

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 for plan bouquets, specially in higher dimensions.

4.6 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.

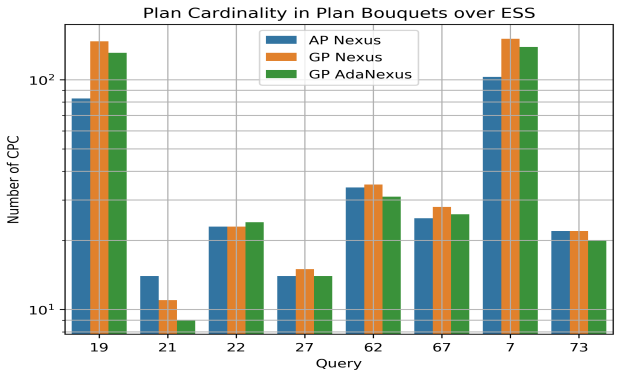


Fig 12. Total contour plan cardinalities

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.

5 Conclusions

Firstly, we have proved (with an additional assumption of Bounded Cost Growth) that exponential distribution in ESS discretization leads to usable cost deviation in contour discovery. We have given upper bounds on cost deviations and validated that they are followed in all our test queries.

Next, we have devised an improvement over NEXUS with complete removal of explicit ESS discretization which leads to contour discovery with lesser cost deviations and also with lesser optimizer calls.

6 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 design 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.

References

[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.

[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.

[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.

[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.

[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

[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

[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.

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

[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.

[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.

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

[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.