Plan Bouquets based Techniques for Variable Sized Databases

M.Tech Project Report

Achint Chaudhary (CSA)

April 18th, 2020

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 to set of plans. This technique provides strong bounds independent of data distribution.

Plan bouquets on cost sub-optimality is not designed to be robust against large updates in the database. This work focuses on observing limits up to which size of database can be increased without serious deterioration in performance guarantee. Also, we will provide incremental algorithms that can use information from plan bouquet compiled in past and extend it to achieve robust execution without incurring overhead of re-compiling entire plan bouquet.

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 will process 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 hypercube. The set of optimal plans over the entire range of selectivity values in ESS is called $Paramteric\ Optimal\ Set\ of\ Plans(POSP)$. 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.

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

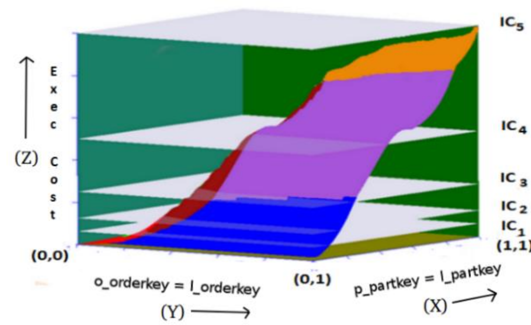


Fig 1. OCS and Plan Trajectories intersection

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.

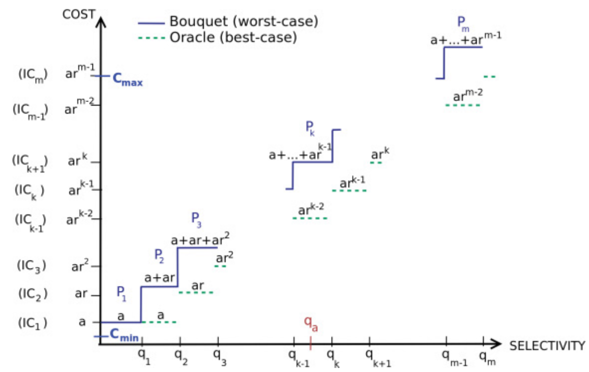


Fig 2. Cost incurred (Oracular vs Bouquet)

In the above figure, 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

$$C_{bouquet}(q_a) = cost(IC_1) + cost(IC_2) + \cdots + cost(IC_k)$$

$$= a + ar_{pb} + ar_{pb}^2 + \cdots + ar_{pb}^{k-1} = \frac{a(r_{pb}^k - 1)}{r_{pb} - 1}$$

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

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

$$= \frac{r_{pb}^2}{r_{pb} - 1} - \frac{r_{pb}^{2-k}}{r_{pb} - 1}$$

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

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 $4 * \rho$, 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] was invented, 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

$$D^2 + 3D$$

We will be using SpillBound in some sections for our work, as it provides pre-compilation performance guarantee based just on query inspection, and also platform independent

2. PROBLEM FORMULATION

2.a Notations

Notation	Description
SP	Selectivity Predicates
WKP	Well Known Predicates
EPP	Error Prone Predicates
TP	Trivial Predicates
ESS	EPP Selectivity Space
OCS	Optimal Cost Surface
$POSP$	Parametric Optimal Set of Plans
RES	Resolution of Discretized ESS
Dim or D	Dimension of ESS
$(1+\Delta)_i$ or S_i	Scaling Factor of Predicate SP_i
$\mathcal{E}_i$	Minimum Selectivity on Predicate SP_i
m	Number of Iso-cost Contours
IC_i	Iso-cost contour with index i
CC_i	Cost Budget of IC_i
r_{pb}	Cost Ratio of Iso-cost contours
$(0, 1.0]$ or $[\mathcal{E}, 1.0]$	Selectivity interval for an axis of Discretized ESS
P_j	Plan with assigned identity j
F_i	Plan Cost Function for Plan P_i
$Cost(P, q)$	Cost of plan P at location q in ESS of reference database

$Card(p, q, scale)$	Cardinality of predicate p at location which has undergone change of $scale$ w.r.t to reference database
d_{sel}	Difference in consecutive selectivity values on axis of ESS
r_{sel}	Ratio of consecutive selectivity values on axis of ESS
β_{max}	Worst case slope of Plan Cost Function
α	Tolerance of contour thickening

2.b Assumptions

2.b.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.b.2 Axis Parallel Concavity (APC)

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

$$F_p(\theta * q_a + (1-\theta) * q_b) \geq \theta * F(q_a) + (1-\theta) * F(q_b)$$

Generalizing to D dimensions, a PCF F_p is said to be $axis\ parallel\ concave$ (APC) if the function is concave along every axis-parallel 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 $PCFs$ 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.b.3 Bounded Cost Growth (BCG)

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

$$F_p(\alpha * q.j) \leq f(\alpha) * F(q.j)$$

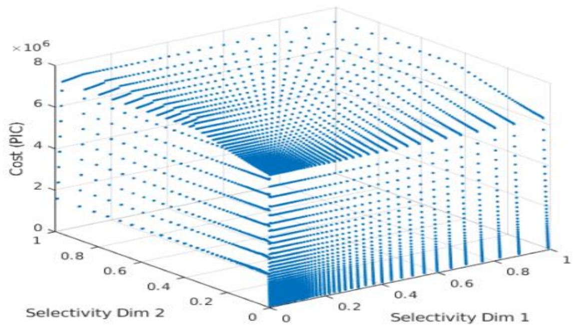
$$\forall j \in \{1,2,\ldots,D\}\ and\ \forall \alpha \geq 1$$

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.

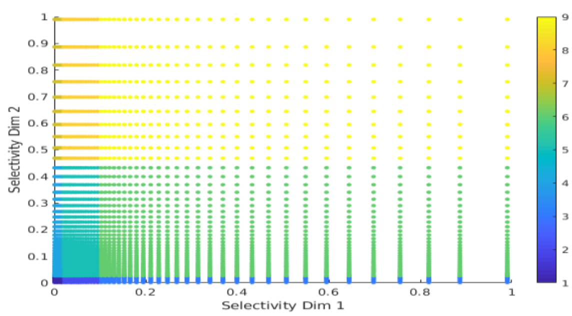
7.7.1.2 Concavity They have also claimed that identity function $f(\alpha) = \alpha$ suffices in practice.

2.b.4 Piecewise Axis Parallel Linear (APL)

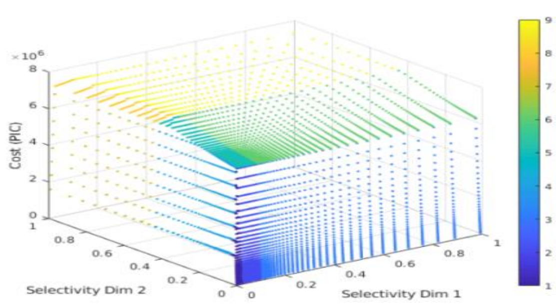
Plan Cost functions and OCS are shown to be piecewise linear in [5]. This property commonly comes from the fact that partial derivatives of common physical operators (except the sort operator, which is seldom found in industry strength benchmark [4]) are linear in nature.



(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 piecewise linear, a coarse approximation of piecewise linear function can still be fitted to them. Similar work has been done in our lab in the past by [6].

While in our work, there is no need to fit any such piecewise linear function. This will reduce our effort of fitting points from entire OCS into a piecewise APL function which will itself be exponential in nature.

2.b.5 EPP are the only predicates

For present analysis, we have considered that all of query predicates are error-prone and there is no trivial predicate which means that each relation has some filter predicate applied over it. This will also be considered as error-prone in our conservative assumption. This will also be

$$EPP \leftarrow SP\ (i.e., WKP \cup EPP \cup TP)$$

Rational behind this assumption is that, if we supply same selectivity value to both old and new database, their outputs will be of different cardinalities. If we have not gone with this conservative assumption, we will need to determine change of scale for WKP and TP, and for now we wish to handle all kinds of predicates in the same picture of analysis.

2.b.6 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.b.7 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.c Performance Metrics

2.c.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 prior:

$$Subopt(q_e, q_a) = \frac{Cost\left(P_{opt}(q_e)\right)}{Cost\left(P_{opt}(q_a)\right)}$$

Here both $q_e, q_a \, \epsilon \, ESS$

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:

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

2.c.2 Worst case Sub-optimality

Worst case sub-optimality is sub-optimality w.r.t to q_e that causes maximum sub-optimality 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.c.3 Maximum Sub-optimality (MSO)

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.d Updated notion of Selectivity Intervals

Selectivity is the fraction of tuples that results from a query predicate. Notation of selectivity is devised to make study of ESS independent of cardinality values. This way selectivity is defined to always lie within bounded interval of [0,1].

Now, we will look at the type of changes in the data stored in a database instance. It can be:

I. Distributional Change
II. Volumetric Change

Plan bouquets and techniques developed later on similar ideas are robust to distributional changes. In the case when only data distribution has changed and all predicates are error-prone, a plan bouquet compiled in the past can be re-used with same MSO_g.

When the size of database increases, the number of tuples to process and maximum possible tuples increases generally.

The choice of optimizer for physical operator is highly dependent on cardinality values. Hence, these choices and resulting plans are also amenable to change.

So, if we go with [0,1] picture of selectivity for both earlier and updated instances. Same selectivity value will result in different cardinality on the database before and after volumetric update.

This difference of cardinality for same selectivity value in both instances will lead to a change in the choice of operators and may result into overall change in the structure of plans. This will ultimately change both, the shape of iso-cost contours as well as the plans lying on them.

The rigid picture of [0,1] selectivity will make the usage of information from contours, generated on the earlier instance for incremental or faster compilation, difficult.

So, across multiple instances of database, same selectivity value should return a same cardinality across multiple instances.

In a loose sense, selectivity to cardinality mapping should not be changed across different sized instances, while we opt to change the selectivity interval for our analysis.

For example, if total tuples in a relation before update are 100, and post update it has become 200. Then, we wish that 0.5 selectivity on both databases should return 50 tuples (this 50 is with reference to earlier instance). So, for initial database selectivity values are from [0,1], while for instance post update, legal selectivity interval is [0,2].

Note that only the base (reference) instance on which plan bouquet is first compiled has selectivity values in [0,1]

This may seem to be fuzzy at first, but we will later show importance of this notation for reducing cost of incremental compilation.

Also, we will see change in volume from ratio of change in maximum cardinality possible from a predicate. This comes from that notion that each axis of ESS denotes a predicate. Hence, a change in the database may cause ESS to grow differently on each axis.

A pictorial representation of this approach is shown in Fig 4. Also, potential regions for which incremental algorithms are needed to be developed are mentioned in the Fig 4.

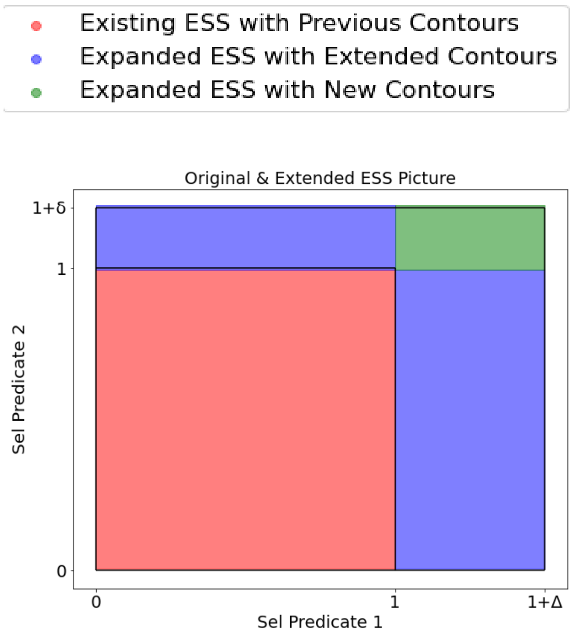


Fig 4. Representation of different regions of concern in cardinality space with new approach of selectivity intervals

In the above diagram, green region is the portion for which standard compilation procedure of contour discovery needs to be called.

For now, we will look for approaches to

I. Extend existing contours which lie at intersection boundaries of red and blue regions into blue region. Note that if a contour will extend in both blue regions above, two seeds will be discovered and two independent NEXUS, one for each blue region will execute to extend each contour.

II. Check the impact of update in cost of points lying on contours with information of their location in ESS and optimal plan at each individual point. This calculation will be helpful to provide relaxed MSO_g from using old contours and plans within red region.

III. If usage of any old contour segment is deteriorating performance guarantee significantly, then incrementally compile contours in red region from algorithms utilizing information from contours built in the past on initial database instance. During incremental compilation, information from past contours can be used to speed up contour discovery.

3 CHALLENGES W.R.T UPDATES

Under the situation of change in database instance, placements of ideal contours can be totally different from the contours built using earlier database instance, which shows that old bouquet contours and plans on them are totally different from new bouquet contours. Under above mentioned conditions it seems that a re-compilation will be needed.

Plan bouquets is suitable for canned queries as compilation overhead of entire discretized ESS enumeration will take $O(RES^{Dim})$ cost, which is amortized over repeated invocations for canned queries.

Now we will look at the way compilation (iso-cost surface identification) is done in the past literature [1], and why under all present options compilation at its first place is resource exhaustive.

3.a Math behind 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-contours (m) using these two values is obtained as follows:

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

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:

$$SubOpt(*, q_a) = \frac{ar_{pb}^{i-1} + ar_{pb}^{i-2} + \cdots + ar_{pb}^{1} + ar_{pb}^{0}}{ar_{pb}^{i-2}}$$

$$\leq \frac{\frac{a\left(r_{pb}^{i} - 1\right)}{r_{pb} - 1}}{ar_{pb}^{i-2}} = \frac{r_{pb}^2}{r_{pb} - 1} - \frac{r_{pb}^{2-i}}{r_{pb} - 1} \leq \frac{r_{pb}^2}{r_{pb} - 1}$$

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, $r_{last} < r_{pb}$. Now using expression for sum of geometric progression, MSO_g for plan bouquets is evaluated as follows:

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

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

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

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

In both case 1 and case 2, we have got same final expression, while in case 2 we have used upper bound for r_{last}. This means that expression from Case 1 provides MSO_g.

When $r_{pb} = 2$ is substituted in the final expression, sub-optimality in that case is also upper bounded by 4.

This same proof can be easily extended for multi-dimensional plan bouquets.

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.

3.b 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:

I. Full ESS enumeration
II. NEXUS.

Let's see them one by one.

3.b.1 Full discretized 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 $O(RES^{Dim})$ 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.b.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 $O\left(m * D * RES^{Dim-1}\right)$.

At first, NEXUS seems to be promising for reducing compilation overhead, but faces following multiple issues [8]:

I. If large number of contours need to be drawn, NEXUS is effectively close to Full space enumeration, especially in high dimensional ESS.
II. 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.
III. 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.c Complexity Issues in Compilation

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

I. Complexity exponential in Dim
II. Need of sufficiently high resolution on each axis

The algorithm with complexity $O(RES^{Dim})$ 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.

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

II. 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.

$while\ (S\ has\ a\ neighbor\ in\ 4^{th}\ quadrant):$

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

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

$$else:$$

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

Algo 1. Neighbor discovery of 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 5.

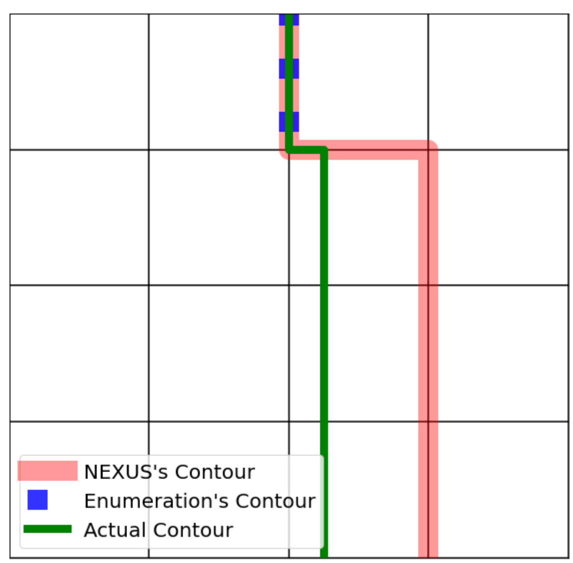


Fig 5. Contour discovery with low resolution and uniform selectivity distribution

To avoid this yet keeping computational feasibility, one possible option is to raise the value of α but that does impact MSO_g . 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_g 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 MSO_g are the missing part from literature which we will try to provide and prove in a systematic way.

4. CONTRIBUTIONS

4.a Increasing compilation efficiency

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

(i) Same effective cost as full ESS enumeration

(ii) 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

$$O(\frac{RES}{m * D})$$

As number of dimensions (D) 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 naïve algorithm.

But before we start working on a improved version of NEXUS, we will first try to look at second argument made against NEXUS.

4.a.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,

$$IC = \{IC_1, IC_2, \dots , IC_m\}$$

$$Q = \{q_1, q_2, \dots , q_m\}$$

Here, $q_1 = 0 \; and \; q_m = 1$

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 suboptimality 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 put on improvement into 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 4th quadrant in its neighborhood. Below example borrowed from [1] shows working of NEXUS in pictorial way in Fig 6.

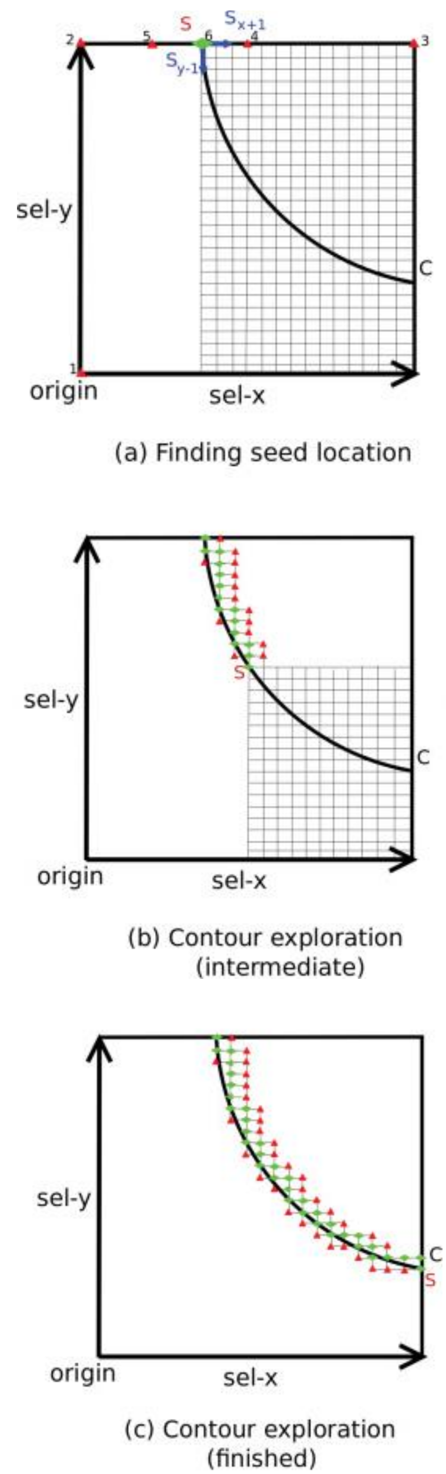


Fig 6. Working of NEXUS

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 bit smart, but we can still improve it using piecewise linear geometry of contours.

From past works [6, 10], contours are observed to be either piecewise linear or approximated to be piecewise linear. See figure for reference of contours generated on a 50GB TPC-DS with Query instance Q91.

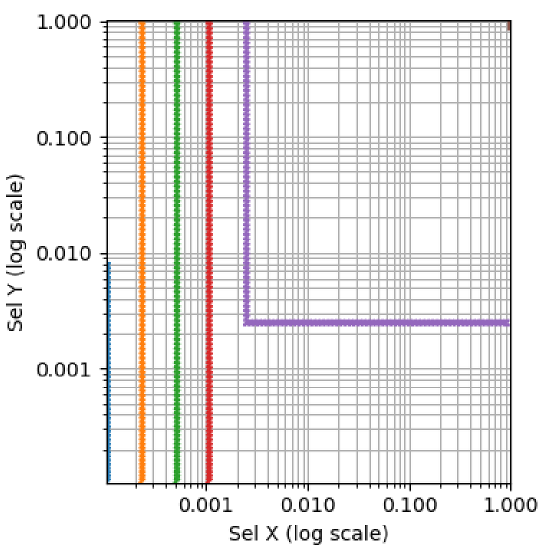


Fig 7. Q91 contours on 50GB TPC-DS

We'll attempt to utilize this highly piecewise linear nature of contours to get improved and faster version of NEXUS, namely NEXUS++, which should in practice speed up the contour discovery process, the main cost overhead of compilation.

We will use same seed discovery process as of NEXUS, using binary search in interval of valid axis, from where we can start the search. Next we'll look at design of contour exploration of NEXUS++.

As an example, consider the red contour (which is 4th 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 $O(RES)$ to $O(\log(RES))$.

There are some fundamental issues with this approach:

(i) Slope in ESS space for any piece of piecewise linear contour is not known a priori
(ii) Even if exact slope can be approximated somehow, 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 $bisection(left, right)$:

$P_L, P_R = P_{opt}(left), P_{opt}(right)$

$if\ (left \leq right)\ and\ (P_L\ != P_R)$:

$mid = \frac{left+right}{2}$

$P_M = P_{opt}(mid)$

$if\ (P_L != P_M)$:

$bisection(left,\ mid)$

$if\ (P_R != P_M)$:

$bisection(mid,\ right)$

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

Using above mentioned procedure within exponential search may not fully reduce optimizer calls for finding all the plans in between plans at end points discovered by exponential search, in the case of large number of plans or alternating plans. First line in Fig 8 shows such a case, on base of which we will develop our concept further.

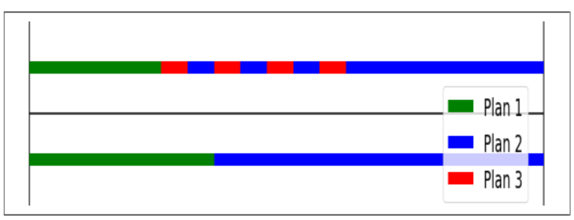


Fig 8. Alternating plans missed by bisection search

Plan bouquet relies on low contour density, which is obtained at intra-contour level using anorexic reduction, once we have obtained all points on Contour. While Spill bound doesn't rely on this reduction due to contour density independent execution.

We will be using a form of plan swallowing in bisection search which reduces the compilation efforts in terms of optimizer calls at expense of some FPC calls. This in practice reduces possibility of alternating with nearly same cost in region between two points discovered by exponential search. This will lead to scenario as shown by second line in Fig 8.

We can modify our last procedure to implement cost swallowing to reduce overheads. Pseudo-code for which is as follows:

Let λ be maximum cost relaxation due to plan swallowing. In practice, 0.2 value is enough as studied in literature [11].

$\psi = (1 + \lambda)$ // Plan substitution threshold

def $bisectionAPD(left, right)$:

$P_L, P_R = P_{opt}(left), P_{opt}(right)$

$if\ (left \leq right)\ and\ (P_L\ != P_R)$:

$mid = \frac{left+right}{2}$

$P_M = P_{opt}(mid)$

$if\ Cost(P_M, mid) * \psi \leq Cost(P_L, mid)$

$if\ (P_L != P_M)$:

$bisection(left,\ mid)$

$if\ Cost(P_M, mid) * \psi \leq Cost(P_R, mid)$

$if\ (P_M != P_R)$:

$bisection(mid,\ right)$

Algo 3. Bisection search with plan swallowing

Now, let's come to the first issue of our search approach, which is getting slope of each piece of piecewise 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 $[(1-\alpha) * CC_i, (1+\alpha) * CC_i]$ and our search method will try to pick the point having optimal cost lying in mid of specified cost interval.

With the knowledge of seed, we will start from one end of contour (one of the many connected linear pieces) and will search in 4th quadrant. Slope information will be obtained on-the-fly and tuned based of 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.

This will not constitute as 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.

Also, a common observation is that, sum of all slope changes of contour will be maximum of 90 degree. In worst case, which is also observed in 5th contour in Fig 7, contour will take a sharp right angle turn anti-clockwise. In this search, no fallback can get us the correct direction.

Once we have detected that fallback strategy will not work, we will go with exponential rotation in 4th contour anti-clockwise to get next correct direction. 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 piecewise linear contours, which is observed in practice.

def $NEXUS++(CC_i)$:

$S_{now},\ P_{now} = InitializeSeed(CC_i)$

$C_{now} = Cost(P_{opt}, S_{now})$

$step,\ \Delta_{now} = 1\,,\ [-1, 0]$

$while\ True$:

$S_{next} = S_{now} + step * \Delta_{now}$

$C_{next} = Cost(P_{opt}\,, S_{next})$

$if\ \max\left(\frac{C_{next}}{CC_i}\,, \frac{CC_i}{C_{next}}\right) \leq (1+\alpha)$:

$\Delta_{now} = TuneDir(S_{now}\,, S_{next})$

$bisectionAPD(S_{now}\,,\ S_{next})$

$S_{now}\,, step = S_{next}\,, 2 * step$

$else$:

$if\ step > 1$:

$step = step\ /\ 2$

$else$:

$\Delta_{now} = RotateDir(S_{now}\,, \Delta_{now})$

Algo 4. NEXUS++ with $bisectionAPD$, 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 $[(1-\alpha)*CC_i$, $(1+\alpha)*CC_i]$ to avoid surface thickening like NEXUS.

$Q-Error$ [12] is chosen as error function to make NEXUS++ algorithm discover points having cost close to mid value of tolerable cost interval. An online tuning algorithm like $PID\ control$ [13] can be used for tuning direction vector.

Note that this slope information is crucial, as for highly piecewise linear contours, having a prior knowledge of slope will dramatically reduce the number of optimizer calls which will otherwise be made during tuning to obtain correct slope.

4.b Geometric progression to discretize each axis of ESS with bounded MSO_g

Earlier full space exploration and NEXUS need 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 two following options:

- (i) Sufficiently high resolution with uniform distribution on each axis
- (ii) Selectivity values should be chosen carefully in geometric progression with bounded relaxation in MSO_g

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 bounded but relaxed MSO_g. Now we will prove the same.

Formula for last term of GP is

$$b = ar^{n-1}$$

Let ε_i be the minimum selectivity to start with on axis corresponding to SP_i.

$$1 = \varepsilon_i * r_i^{RES_i - 1}$$

From slope bounded cost growth, we know

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

Combining the above two equations and re-arranging will yield

$$\varepsilon_i \geq \left(\frac{\beta_i}{r_{cost}}\right)^{RES_i - 1}$$

All four variables in resulting inequality are hyper-parameters to deploy ESS construction in practice. We will see impact of above formulation on MSO_g and meaning of each of these variables.

Variable	Interpretation
ε_i	First selectivity value to start forming GP
RES_i	Resolution of ESS along axis of SP_i
β_i	Maximum slope of OCS w.r.t SP_i
r_{cost}	Ratio of cost change upon each step on selectivity axis of ESS

We will soon see a approach to find ε_i and β_i. For now, if we have these two values and given $r_{cost}(> \beta_i)$, we can get the value of RES_i to build a geometric progression of selectivity value along the axis of SP_i in ESS as

$$RES = \left\lceil \log_{\left(\frac{\beta_i}{r_{cost}}\right)}(\varepsilon_i) \right\rceil + 1$$

With a bounded relaxation factor of η in MSO_g and a total of Dim dimensions in ESS. We will use

$$r_{cost} = \sqrt[Dim]{\eta}$$

This is the maximum cost jump on each axis with use of geometric progression. Like proof of Frugal-SpillBound [3], we can prove that with use of Geometric progression to construct ESS, relaxed MSO_g using plan bouquet will be

$$4 * \rho * \eta$$

in case of SpillBound, MSO_g will become

$$(D^2 + 3D) * \eta$$

Now we will investigate computation of ε_i and β_i.

Calculation of $\hat{\varepsilon}$ for each SP_i

Let $\hat{0}$ be Dim-dimensional vector containing absolute zero selectivity values for each predicate. Also, let $\hat{\varepsilon}$ be the vector we are looking for to construct geometric progression of selectivity values along each axis of ESS.

We will choose $\hat{\varepsilon}$ by doing binary search along diagonal connecting $\hat{0}$ *and* $\hat{1}$ Such that

$$\frac{Cost(\hat{\varepsilon})}{Cost(\hat{0})} \leq MSO_g$$

Since MSO_g computation before compilation is possible in SpillBound, we will use SpillBound in all our experiments

$\hat{\varepsilon}$ is chosen to be new origin and terminus will be $\hat{1}$ as always during compilation.

This method provides $\hat{\varepsilon}$ for forming geometric progression on each axis of ESS and MSO_g for queries lying in $[\hat{0}, \hat{\varepsilon})^D$.

Calculation of $\hat{\beta}$ for each SP_i

$\hat{\beta}$ can be calculated in close neighborhood of $\hat{\varepsilon}$ by varying the value of each SP_i one at a time. Due to concavity assumption, highest slope will lie at origin $\hat{\varepsilon}$.

Till now, we have suggested empirical and algorithmic suggestions for gaining speed up in the compilation process itself.

Now we will look specifically at techniques which can handle incremental database instances in plan bouquet-based algorithms.

4.c Computation of Inflated MSO

4.c.1 Exact inflated MSO_e for updated ESS

At first, we will look at a simple procedure to compute nearly accurate value of empirical guarantee value (MSE_e) under assumption of perfect cost model.

We can choose all points in ESS, and simulate our robust algorithms (under the assumption of perfect cost model, simulation can be done in parallel), and can find out maximum relaxed MSO_e. Complexity for this procedure will be

$$O(\prod_{i=1}^{Dim} RES_i) = O(RES^{Dim})$$

4.c.1 Inflated MSO_g for updated ESS.

With use of NEXUS++ and ESS discretized using geometric distribution of selectivity values on each axis, we will obtain substantially lesser number of points to represent each contour, as compared to number of points obtained with classic NEXUS and uniform distribution of selectivity values on each axis in a high resolution picture.

In the combined (existing and extended) regions of ESS after database scales up, we will use these points with FPC module to obtain inflated MSO_g . Inflation in MSO_g can be observed when we choose to continue using old representative points of contours from existing region of ESS. Note that in the extended ESS, we will always go with standard compilation procedure.

Now, we will look at an algorithm for efficient computation of MSO_g. Consider that each contour IC is a collection of tuples (q, P) where q is the location on contour and P is the plan at that point available on IC contour.

$r_{max} = r_{min} = r_{pb}$

$for \quad ix \in [1, m-1]:$

$\quad early, next = ix, ix+1$

$$\text{early}_{\min} = \min_{(q,P)\in IC_{early}} \big(Cost(P,q)\big)$$
$$\text{early}_{\max} = \max_{(q,P)\in IC_{early}} \big(Cost(P,q)\big)$$
$$\text{next}_{\min} = \min_{(q,P)\,\in\, IC_{next}} (Cost(P,q))$$
$$\text{next}_{\max} = \max_{(q,P)\,\in\, IC_{next}} (Cost(P,q))$$

$$r_{inflation} = \left(\frac{next_{max}}{early_{min}}\right)$$

$$r_{suppression} = \left(\frac{next_{min}}{early_{max}}\right)$$

$\quad if\ r_{inflation} > r_{max}:$

$$r_{max} = \left(\frac{next_{max}}{early_{min}}\right)$$

$\quad if\ r_{supression} < r_{min}:$

$$r_{min} = \left(\frac{next_{min}}{early_{max}}\right)$$

$$MSO_g = \max\left(MSO(r_{max}), MSO(r_{min}) \right)$$

Algo 5. Computation of Inflated MSO_g

This method of computation of MSO_g does not involve any simulation of plan bouquet-based algorithm and will only include FPC calls equal to the sum of number of points on all contours.

The way we are looking to extend ESS upon volumetric updates and try to use fixed set of contours, one thing to note is that C_{min} will remain same across different instances. On the other hand, ratio of selectivity growth r_{sel} may increase when moving across axis into extended region.

So, like the way we computed inflated MSO_g, we can compute order of contours needed to be re-constructed to lower down inflated MSO_g close to ideal value of $4 * \rho$ or more precisely to $D^2 + 3D$ with SpillBound.

Following procedure finds out contour causing max inflation in MSO_g using greedy approach and should be re-built. This contour will be built using incremental NEXUS++ resulting in less inflated MSO_g.

def $GCC(Contour\ List)$:

$IC_{redo}\ , r_{change} = None\ , 1$

$for\quad IC_{ix} \in Contour\ List:$

$$CC_{\min} = \min_{(q,P)\in IC_{ix}} \left(Cost(P,q)\right)$$

$$CC_{\max} = \max_{(q,P)\in IC_{ix}} \left(Cost(P,q)\right)$$

$$r_{deviation} = max\left(\frac{CC_{max}}{CC_{ix}}, \frac{CC_{ix}}{CC_{min}}\right)$$

$if\ r_{deviation} > r_{change}:$

$IC_{redo} = IC_{ix}$

$r_{change} = r_{deviation}$

$return\ IC_{redo}$

Algo 6. Greedy Contour Construction (GCC)

4.d Incremental bouquet maintenance

Till now, we have added efficient methods for compilation, relaxed MSO_g computation and choice of contours to re-draw. Another issue that has not been discussed yet is as follows.

A database may get multiple volumetric updates over time. On each update, we will extend existing contours and draw few new ones. This avoids re-compilation of entire bouquet on each update.

For current update, some part of existing ESS from first compilation may lead to serious deterioration in MSO_g while impact from rest of the parts can be tolerated. A sample instance of this is shown in following Fig 9.

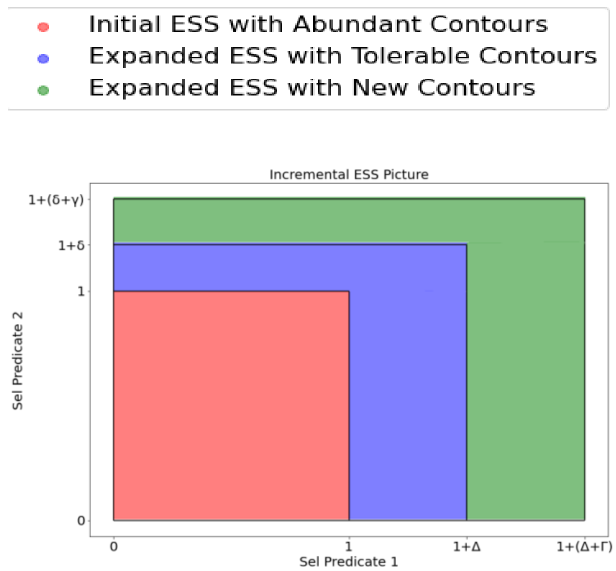


Fig 9. Possibility on variety of regions from extended ESS into constituting towards MSO_g upon successive updates

The contours lying in all these regions share same C_{min} and r_{pb} and hence cost values of contours can be exactly same between two regions of ESS (named sub-ESS). Contours lying in each region can be maintained independent of each other (for MSO_g impact and re-drawing using GCC algorithm). Contours with same cost value lying in different sub-ESS need only be merged for execution of bouquet algorithm.

So, in the above figure we can selectively re-draw contours in the initial ESS shown in red color.

This approach enables us to avoid re-compilation on entire existing ESS but sub-ESS impacting MSO_g. It also suggests that each sub-ESS should be maintained independent of each other.

4.e Constant Selectivity to Cardinality mapping

We will now make another clear rationale to choose a selectivity notion not bounded within $[0, 1]$.

Cost based query optimizer chooses physical operators for each logical operator in abstract relational algebraic tree. Cost of multiple physical operators will be compared in two ways:

I. Absolute values
II. Relative values

Cost of operators used for base relation filter predicates are compared in a relative manner while operators for join are generally compared in absolute manner. Tipping point of these decisions are cardinality values where the cost of both the choices on either side is same.

To observe importance of this notion of selectivity interval outside $[0, 1]$, let's view it with an example.

Suppose, we have an SQL query with only single operator (corresponding to an EPP) where decision about choice of operator must be made. Let's make another assumption that this choice is based on absolute cost. Then the decision will be like:

$if\ Cost(Operator_A) \leq\ Cost(Operator_A)$:

$\quad Use\ Operator_A\ in\ Query\ Plan$

$else$:

$\quad Use\ Operator_B\ in\ Query\ Plan$

Since the decision is absolute, tipping cardinality value will remain same no matter what maximum cardinality is possible for that selectivity predicate. Fig 10. given below pictorially represents the same.

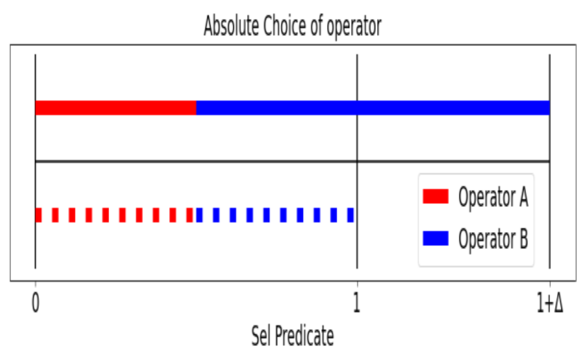


Fig 10. Dependency of absolute choice of operator on data volume to process

Dashed line shows the scale of data to process with different colors for each operator. Transition between two colors is the tipping point. Since decision is absolute, same cardinality value will remain tipping point even after database update, which is shown as solid line for increased volume.

Suppose that for a SQL query, all predicates are error-prone (all data flowing into query execution plan has entered from some EPP), all choices are absolute and optimizer has perfect cost model based on cardinalities only, then choice of operator in [0, 1] interval will never change for each predicate. After all plan structure is just a arrangement of operator within tree, if cost of no operator is changing, overall plans in ESS compiled on initial instance of database will remain same, leaving us only to compile contours in extended ESS. This is a picture which is too good to be true.

All choices are not absolute, most base relation filter predicates are relative in nature and tipping point of decision will change as depicted in figure below.

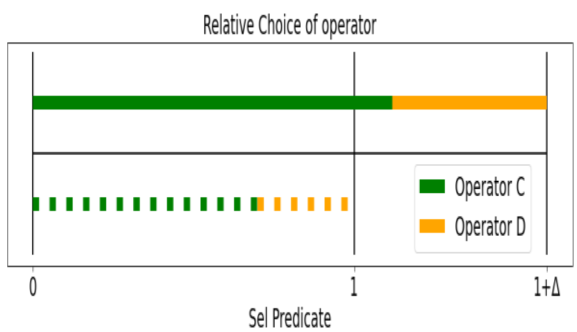


Fig 11. Dependency of relative choice of operator on data volume to process

$$if\ \frac{Cost(Operator_A)}{Cost(Operator_B)} \leq\ \delta :$$

$$Use\ Operator_A\ in\ Query\ Plan$$

$$else :$$

$$Use\ Operator_B\ in\ Query\ Plan$$

These relative choices will need one to re-compile (reconstruct contours) in some places within existing ESS also.

Now as we know that optimizer choices are totally cardinality based but can be both absolute and relative. Where relative choices in lower part of query plan may also affect absolute choices to be made in upper part of query plan and vice versa.

We will exploit a basic idea that tipping point for absolute choice-based operators will never change. Also, for operators having a relative choice, the tipping point between the first two choices of physical operators in decreasing order of their cost function complexity will always move to high cardinality value from the previous tipping point.

So, there will always be a region of ESS where no operator decision will change ultimately leading to same optimal plan, no matter how much volumetric update happens in the database instance. Such a region is known as $Safe - Sub - ESS$.

All contours lying completely in Safe-Sub-ESS will be called $static\ contours$ and never need to be re-compiled or even checked during inflated MSO_g computation.

The optimal plans and cost values of the points on those contours will never change.

Note that volume of Safe-Sub—ESS as compared to entire ESS depends on number of choices for implementing physical operators for each logical operator. While all platforms have variety of implementations, typical number for each logical operator still lies within 5.

4.f Finding $Static\ contours$

As we know that all points on any contour IC_k will collectively dominate all contours from IC_1 to IC_{k-1}. So, contours should be marked static in increasing order of their cost budget.

A necessary condition for a contour to be static is that it should lie completely within existing ESS. ESS and cost of any point on static contour should not change. This check using FPC will act as sufficient condition.

def $CheckStatic(IC)$:

  $for\ (P,q) \in IC$:

    $if\ Cost_{new}(P,q) \mathrel{!=} Cost_{old}(P,q)$:

      $return\ False$

    $return\ True$

def $MarkStatic(DynamicContours)$:

  $for\ IC_i \in DynamicContours$;

    $if\ IC_i.points \nsubseteq ExistingESS$:

      $break\ Loop$

    $bool_{static} = CheckStatic(IC_i)$

    $if\ bool_{static}$:

      $StaticContours.add(\ IC_i\ )$

    $else$:

      $break\ Loop$

Algo 7. Pseudo-code for Marking static contours

4.g Using existing contour geometry

The cost associated with optimizer calls is dominated by the tuning of slope value with exponential search.

In this section, we will see an idea and corresponding algorithm that will pass available geometric information from existing contour which need to be re-drawn using NEXUS++, so that lesser optimizer calls should results, which are otherwise spent on tuning of slope to guide exponential search used in discovery of points on contour. If the geometric information is of no use for next contour, it will just result into more optimizer calls as if NEXUS++ is working from scratch without incurring any harm to MSO_g.

We will initialize two seeds, instead of one, and start search of our NEXUS++ from both these ends, taking one step at a time.

Slope information from both ends of previous contour is fed into NEXUS++. It will return two NEXUS++ instances and both of which will act as iterators from which points will be drawn in interleaving fashion.

Among these two points, one initialized from starting seed will move forward, while another initialized from last seed value will move in reverse direction. It must also be checked that point searched from end should always lie in 4th quadrant of point searched from beginning. If this condition is ever violated, a simple bisection search will work for all points between them.

def $BiNEXUS++(IC_i, CC_i)$:

$S'_S, S'_L = Start(CC_i), Last(CC_i)$

$Iterator_s = NEXUS++(CC_i, IC_i, S'_S)$

$Iterator_L = NEXUS++(CC_i, IC_i, S'_L)$

$while\ |S'_S.x - S'_L.x| \wedge |S'_S.y - S'_L.y|$:

$step_S = Iterator_S.next()$

$if\ S'_S + step_s \leq S'_L$:

$S'_S = S'_S + step_s$

$else$:

$break\ Loop$

$step_L = Iterator_L.next()$

$if\ S'_S \leq S'_L - step_L$:

$S'_L = S'_L - step_s$

$else$:

$break\ Loop$

$if\ |S'_S.x - S'_L.x| \vee |S'_S.y - S'_L.y|$:

$bisectionAPD(S'_S, S'_S)$

Algo 8. BiNEXUS++ for using information of existing contour geometry

4.g End-to-End Incremental bouquet

I. Find new C_{max} post update and scale of selectivity change for each error-prone predicate
II. Check if any sub-ESS has any contour missing. If so, draw missing contours in that sub-ESS.
III. Compute inflated MSO_g . If MSO_g is tolerable, halt incremental compilation procedure
IV. Find contour among non-static contours in existing ESS that should be redrawn to reduce MSG_g
V. Reconstruct the selected contour using geometric information from past contour
VI. Detect and mark static contours
VII. Repeat Step II.

5 EXPERIMENTS

6 CONCLUSIONS

We have first reduced compilation overheads from what is done in past literature.

We have given a proof on usage of geometric progression to discretize axes with bounded relaxation in MSO_g to eliminate the need of high-resolution uniform distribution of selectivity values even for classical methods.

Later we have shown efficient methods to determine inflated MSO_g using information about contours from old and extended ESS.

We have also provided incremental bouquet maintenance algorithms and framework.

No such improvement seems possible with $SchematicReduction$. One thing to note is that $WeakDimRemoval$ is a technique applicable post compilation and can be incorporated in our framework for incremental bouquet maintenance also.

7 FUTURE WORK

7.a Dimensionality Reduction

Since we have used conservative assumption that all query predicates are error prone, this will pose issue for $SchematicRed$ and $MaxSelRemoval$ provided in [6]. Plan reuse from previous compilation may lead to inflated MSO_g as the volume of data input into a query plan via predicates corresponding to removed dimensions could also change upon updates in the database.

One basic improvement in $MaxSelRemoval$ can be to compute $MaxInflationFactor$ with some expected future update in all predicates and compute on corners of $E[Extended\ ESS]$ instead of $[\,\varepsilon\,,1\,]^D$. Or to add a removed dimension later in future with $[\,1\,,1+\Delta\,]$ selectivity interval, unlike an impactful predicate having selectivity interval of $[\,\varepsilon\,,1+\Delta\,]$.

But in that case, ESS would not be a regular D-dimensional hypercube but will have varying number of dimensions. When inflation factor degrades MSO_g, these ξ dimensions can be added back in some regions of ESS making extended ESS to have $D+\xi$ dimensions.

7.b Selectivity Independence

While most work in literature has assumed predicate selectivity independence, it is not actually the case and seldom holds in practice. Initial work to relax this assumption is done by [12].

This assumption can be relaxed at contour level within SpillBound. Instead of assigning budget of CC_i for each predicate selectivity discovery independently. We can check if selectivity discovery for k predicates can be explored in combined way with a budget of $k * CC_i$.

This may empirically reduce the number of $Repeat\ Executions$ which is a dominating contributing factor in the MSO_g proof of SpillBound algorithm but will never deteriorate empirical performance.

We will leave this idea as future work to develop an algorithm to merge predicate selectivity discovery with combined budget, and if possible, to give a proof for improved MSO_g if selectivity dependence functions can be formulated a priori with some domain knowledge.

8 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.cds.iisc.ac.in/publications/report/TR/TR-2018-02.pdf

[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. 2007. On the production of anorexic plan diagrams. In Proc. of the 33rd Intl. Conf. on Very Large Data Bases (VLDB'07). 1081–1092

[10] 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, from https://en.wikipedia.org/w/index.php ?title=PID_controller

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