Dimensionality Reduction Techniques for Bouquet Based Approaches

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Computer Science and Engineering

BY

Sanket Purandare

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

July, 2018

Declaration of Originality

I, **Sanket Purandare**, with SR No. **04-04-00-10-42-16-1-13489** hereby declare that the material presented in the thesis titled

Dimensionality Reduction Techniques for Bouquet Based Approaches

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2016-2018**.
With my signature, I certify that:

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

Date: Student Signature

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

Advisor Name: Prof. Jayant R. Haritsa Advisor Signature

DEDICATED TO

My Family and Advisor

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Jayant Haritsa for providing me an opportunity to work with him. Meticulousness in every action or thought and being completely honest with oneself are some of the most critical teachings he has imparted to me. Zeal for research and hunger to succeed is what I take along with me in future.

I would also like to thank CSA department and Indian Institute of Science for providing me an excellent platform and research facilities. Overall it has been a wonderful learning experience for me.

I would also like to thank Srinivas Karthik for guiding me in my research journey. Finally I would like to thank my family for their unilateral support, my lab mates for their valuable inputs and constructive criticism and my friends for supporting me during critical times.

Abstract

To address the classical selectivity estimation problem in database systems, a radically different query processing technique called `SpillBound` (`SB`) was proposed in [1]. In this approach, the selectivity estimation process is completely abandoned and replaced instead with a calibrated selectivity discovery mechanism. The beneficial outcome is that provable guarantees are obtained on worst-case execution performance, thereby facilitating robust query processing.

Specifically, `SpillBound` delivers a worst-case multiplicative performance bound of D^2+3D, where D is simply the number of error-prone predicates in the user query. But its guarantees are predicated on expending enormous pre-processing efforts during query compilation which are exponential in D. With the performance bound and compile time effort being quadratic and exponential functions in D respectively and the conservative assumption of `SB` that all the predicates in the query are error prone, it falls prey to the curse of dimensionality.

There are queries wherein all the dimensions may not be error-prone or may not equally impact the query processing cost. When a dimension is removed, we lose information regarding its selectivity that may lead to sub-optimal plan choices and subsequently an inflation to the worst-case performance guarantees. In this work we present a two step process, we first present techniques for removal of carefully chosen dimensions resulting in tremendous reduction in compile-time efforts without worsening the initial performance guarantees. We then present another algorithm which improves the worst-case execution performance bound of the query after dimensionality reduction by making it a function of impactful dimensions only.

We have evaluated our techniques on the TPC-DS decision support benchmark and were able to successfully able to bring down the dimensionality of most of the queries from as high as twelve to less than six. Also we were able to improve their performance guarantees substantially.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cost-based database query optimizers estimate a host of selectivities while identifying the ideal execution plan for declarative OLAP queries. For example, consider the simple SPJ query shown in Figure 1.1, here the optimizer estimates the selectivities of a filter predicate (p_retailprice) and two join predicates (part $\bowtie$ lineitem, lineitem $\bowtie$ orders). Predicate selectivity estimates for optimizing OLAP queries often differ from those actually encountered during query execution, leading to poor plan choices and thereby inflated query response times. To address this selectivity estimation problem, a radically different approach called `SpillBound`(`SB`) was recently proposed in [1], wherein the estimation process is completely abandoned and replaced with a calibrated discovery mechanism. The `SB` approach proves that its construction results in bounded overheads for the selectivity discovery process and consequently, guaranteed worst-case performance.

They use the notion of *Maximum Sub-Optimality* (**MSO**), introduced in [2], as a measure of the robustness provided by a query processing technique to errors in predicate selectivity estimation. Specifically, given a query, the MSO of the query processing algorithm is the worst-case ratio, over the entire selectivity space, of its execution cost with respect to the optimal cost incurred by an oracular system that magically knows the correct selectivities. To make the report self-contained we now describe the `SB` Algorithm in brief.

1.1 `SpillBound` Algorithm

In the `SB` technique, a multi-dimensional *Error-prone Selectivity Space* (`ESS`) is constructed at query compile-time, with each dimension corresponding to one of the error-prone predicate selectivities in the query, and ranging over $(0, 1]$. A sample 2D `ESS` is shown in Figure 1.2 for the example TPC-H [8] query of Figure 1.1, where the two join predicates are viewed to be the

problematic error-prone selectivities.

```
SELECT distinct o orderdate FROM lineitem, orders, part WHERE p_partkey =
l_partkey and o_orderkey = l_orderkey and p_retailprice < 1000
```

Figure 1.1: Example TPC-H Query

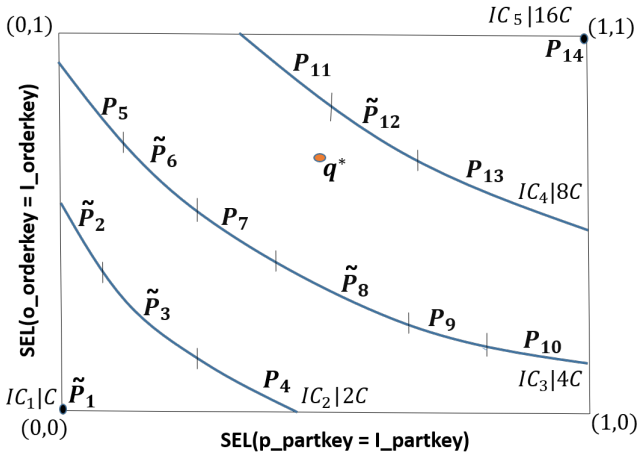


Figure 1.2: SpillBound Execution on 2D ESS

On this ESS space, a series of *isocost* contours, $\mathcal{IC}_1$ through $\mathcal{IC}_m$, are drawn – each isocost contour $\mathcal{IC}_i$ has an associated optimizer estimated cost $\texttt{CC}_\texttt{i}$, and represents the connected selectivity curve along which the cost of the optimal plan is equal to $\texttt{CC}_\texttt{i}$. Further, the contours are selected such that the cost of the first contour $\mathcal{IC}_1$ corresponds to the minimum query cost C at the origin of the space, and the cost of each of the following contours is *double* that of the previous contour.

Therefore, in Figure 1.2, there are five hyperbolic contours, $\mathcal{IC}_1$ through $\mathcal{IC}_5$, with their costs ranging from $\texttt{CC}_1 = C$ to $\texttt{CC}_5 = 16C$.

The union of the plans appearing on all the contours constitutes the "plan bouquet" for the query – accordingly, plans P_1 through P_{14} form the bouquet in Figure 1.2. Given this set, the SpillBound algorithm operates as follows: Starting with the cheapest contour $\mathcal{IC}_1$, a carefully chosen *subset* of plans on each contour are sequentially executed *with a cost budget equal to the contour's cost*. Each plan is executed in "spill-mode" during the discovery process with focus on maximally learning the selectivity of a specific error-prone predicate within its allocated time budget. This process of contour-wise plan execution ends when all the selectivities in the ESS

have been fully discovered. Armed with this complete knowledge, the genuine optimal plan is now identified and used to finally execute the query to completion.

To make the SB methodology concrete, consider the case where the query happens to be actually located at q^*, in the intermediate region between contours $\mathcal{IC}_3$ and $\mathcal{IC}_4$, as shown in Figure 1.2. Assume that the optimal plan for this location, P_{q^*}, would cost $7C$ to process the query. In contrast, SB, which is unaware of the true location, would invoke the following budgeted execution sequence:

$$P_1|C, P_2|2C, P_3|2C, P_6|4C, P_8|4C, P_{12}|8C, P_{q^*}|7C$$

where the initial executions help to determine the location of q^*, and the final P_{q^*} is the ideal plan used to execute the query to completion. (For ease of visualization, the chosen subset of plans in each contour are annotated using the $\tilde{}$ symbol in Figure 1.2).

In the above scenario, the cumulative execution cost incurred by SpillBound is $(C + 2C + 2C + 4C + 4C + 8C + 7C) = 28C$, whereas the oracular optimizer, which magically knows the q^* location, completes in $7C$. This results in a sub-optimality ratio for SpillBound of $28C/7C = 4$ for the q^* location.

The additional execution costs entailed by SB's "trial-and-error" selectivity discovery exercise can be *bounded* relative to the optimal, *irrespective of the query location in the space*. The MSO of SB is bounded by

$$MSO_{SB} \leq D^2 + 3D \tag{1.1}$$

where D is the dimensionality of the ESS, i.e. the expected number of error-prone predicates in the input query.

Limitations of SpillBound Notwithstanding SpillBound's unique benefits with regard to robust query processing, a major limitation is that its MSO guarantees are predicated on expending enormous pre-processing overheads during query compilation. Specifically, identifying the isocost contours in the ESS, entails in principle, $\Theta(r^D)$ calls to the query optimizer, where r is the resolution (i.e. discretization granularity) along each dimension of the ESS. Also the MSO grows quadratically with D. So, for instance, if $r = 100$, corresponding to selectivity characterization at 1% intervals, and D is 10, *100 quintillion* optimizer invocations have to be carried out to identify the contours before SB can begin executing the query and the worst-case performance bound is as high as *130*.

1.2 Dimensionality Reduction Techniques

Currently, the `SpillBound` approach does not provide any systematic way of choosing the error-prone predicates in the query. It conservatively assumes that *all* the predicates in the query are error-prone. This could be a costly assumption to make given that the compile-time complexity of the algorithm is a function exponential in D and the worst-case performance bound is a quadratic function in D. There are queries wherein all the dimensions may not equally contribute to the query processing cost and thereby unnecessarily contributing to the increase in the dimensionality of the ESS leading to higher compile time efforts and inflated worst-case performance guarantees.

In this work we present algorithms which provide a way to analyze the impact of a dimension and deterministically calculate the worst-case penalty incurred by the removal of that dimension from the ESS. After this analysis, a careful choice of the dimensions to be removed is made such that their removal does not worsen the initial MSO guarantee. It is important to note here that removal of a dimension from the ESS causes loss of information related to its selectivity and thereby sub-optimal plan choices. But, if this increase in sub-optimality is low and bounded, such that the resulting MSO after the removal of dimension is less than the initial MSO then the dimension can be removed safely. Random removal of certain dimensions may even cause the MSO to worsen since loss of their selectivity information from ESS may lead to highly sub-optimal plan choices thereby defeating the purpose of their removal, such dimensions are to be retained. Hence the choice of the dimensions to be removed is particularly important.

Evaluation of our techniques on TPC-DS benchmark shows that we are able to remove significant number of dimensions from most of the queries, specifically we are able to bring down the dimensionality of most queries from as high as 12 to less than or equal to 5 as presented in Section 5. This results in exponential savings in the compile time effort. These techniques require minimal computational efforts in comparison to the savings they do as we will see later. After the dimensionality reduction phase we then apply our algorithm for improving MSO guarantees to make them tighter.

The outline of our techniques is shown in Figure 1.3. The first two techniques attempt to reduce the ESS dimensionality without worsening the initial MSO guarantees while the latter tries to improve the MSO guarantees once the dimensionality reduction is done. Let us look at each of the techniques briefly with the help of an example. Consider the TPC-DS [9] Query 27 shown in Figure 1.4.

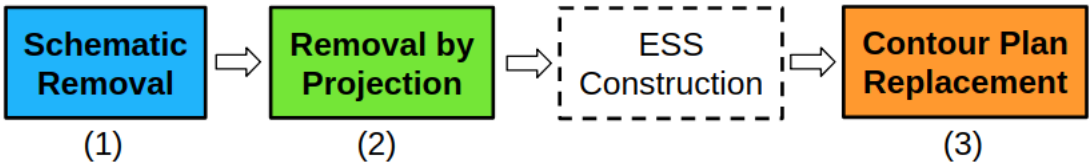


Figure 1.3: Outline of Techniques

```
SELECT i_item_id, s_state, ss_quantity, ss_list_price, ss_coupon_amt,
ss_sales_price FROM store_sales, date_dim , item , store, cus-
tomer_demographics WHERE
ss_item_sk = i_item_sk and ss_store_sk = s_store_sk and
ss_cdemo_sk = cd_demo_sk and ss_sold_date_sk = d_date_sk and
cd_gender = 'F' and cd_marital_status = 'D' and
cd_education_status = 'Advanced Degree' and
d_year = 2000 and s_state in ('TN') and i_current_price ≤ 100
```

Figure 1.4: TPC-DS Query 27

Schematic Dimension Removal : In the Schematic Dimension Removal Scheme we try to make use of the metadata statistics and physical schema to conclude if we can estimate the selectivities of some predicates with high accuracy. For instance, the column '*cd_gender*' which has only two possible values, exact frequency counts for the data are usually maintained and hence its selectivity can be estimated accurately.

For the predicates, for whom the exact frequency counts are hard to maintain due to large domains, we can access the index on those columns (if it exists) to make accurate selectivity estimates for them.

Dimension Removal via Projection : The Projection technique calculates the maximum penalty (called as `InflationFactor`) that can be incurred to MSO if we project the dimension to be removed to its maximum possible selectivity value. The predicates, which in spite of this `InflationFactor`, do not degrade the original MSO can be removed before the construction of ESS and their selectivities can be safely (MSO is safe from being deteriorated) assumed to be 1 for ESS construction. We call this maximum relative increase as `MaxInflationFactor` and show how it can be calculated deterministically and time-efficiently in Section 3.2.

To summarize, in the above query we are able to remove 3 predicates using Schematic re-

moval (coloured blue), 3 predicates using Projection (coloured green) resulting in the removal of 6 out of 10 predicates at compile time with a `MaxInflationFactor` (`MIF`) of 1.17. The direct consequence of this is, the compile time effort reduces from $(res)^{10} \rightarrow (res)^4$. If resolution (res) is 100 then we are *1 trillion* times faster in generating the reduced ESS.

Contour Plan Replacement : After removal of predicates by projection and schematic techniques, we are left with the predicates that have significant impact on MSO if we allow the optimizer to make selectivity estimates for them. In this technique with the help of the `MIFs` calculated earlier, we try to *piggyback* the execution of "weak" predicates (dimensions) along with the "strong" dimensions. The idea is to make the worst-case performance guarantees only a function of strong dimensions modulo the inflation (λ) incurred for *piggybacked executions*.

The Contour Plan Replacement along 2 dimensions (colored orange) in Figure 1.4 causes the MSO to drop from 130 for 10 epps to 17.03 with only 2 impactful dimensions (colored red) remaining in effect.

Table 1.1 summarizes the performance bounds and compile-time efforts achieved by the above techniques when k dimensions are removed using the Schematic and Projection Removal techniques and Contour Plan Replacement is done along m dimensions.

Technique	**No. of Dimensions.**	**MSO**	**Compile Time Effort**
SpillBound	D	$D^2 + 3D$	res^D
Schematic and Projection Removal	$D - k$	`MIF` $* ((D-k)^2 + 3(D-k))$	$res^{(D-k)}$
Contour Plan Replacement	$D - k - m$	`MIF` $* \lambda *$ $((D-k-m)^2 + 3(D-k-m))$	$res^{(D-k)}$

Table 1.1: Summary of Dimensionality Analysis and Removal Techniques

We hasten to add that the MSO after schematic and projection removal of dimensions should be less than the initial MSO for the dimension removal to be safe with respect to MSO. The same applies to MSO after contour plan replacement as well.

Chapter 2

Problem Framework

The problem framework is adopted from [1] since it is identical for the problem being addressed and to make the document self-contained.

2.1 Error-prone Selectivity Space (ESS)

For a query having D error-prone predicates (epps), let the set of error-prone predicates be denoted by $EPP := \{e_1, ..., e_D\}$, where e_j denotes the j^{th} epp. The selectivities of the D epps are mapped to a D-dimensional space with the selectivity of e_j corresponding to the j^{th} dimension of the space. Now, the selectivity of each epp ranges over $(0, 1]$, the result of which is a D-dimensional hypercube $[0, 1]^D$. This is referred to as the Error-prone Selectivity Space (ESS). In practice, an appropriately discretized grid version of $[0, 1]^D$ is considered as the ESS. Note that each location $q \in [0, 1]^D$ in the ESS represents a specific query instance where the epps of the user query happen to have selectivities corresponding to the location coordinates of q. Accordingly, the selectivity value on the j^{th} dimension is denoted by $q.j$.

The notion of a location q_1 dominating a location q_2 in the ESS plays a central role in our framework. Formally, given two distinct locations $q_1, q_2 \in ESS$, q_1 dominates q_2, denoted by $q_1 \succeq q_2$, if $q_1.j \geq q_2.j$ for all $j \in 1, ..., D$. In an analogous fashion, other relations, such as $\succ$, $\prec$, and $\preceq$ can be defined to capture relative positions of pairs of locations.

2.2 Search Space and Cost

We assume that the query optimizer can identify the optimal query execution plan if the selectivities of all the epps are correctly known. Therefore, given an input query and its epps, the optimal plans for all locations in the ESS grid can be identified through repeated invocations of the optimizer with different epp selectivity values. The optimal plan for a generic selectivity

location $q \in ESS$ is denoted by P_q, and the set of such optimal plans over the complete ESS constitutes the Parametric Optimal Set of Plans (POSP) [3]. We denote the cost of executing an arbitrary plan P at a selectivity location $q \in ESS$ by $Cost(P, q)$. Thus, $Cost(P_q, q)$ represents the optimal execution cost for the selectivity instance located at q. Throughout the report, we adopt the convention of using q_a to denote the actual selectivities of the epps in the user query note that this location is unknown at compile-time. Also we denote the deterministic sequence pursued for a query instance corresponding to q_a by Seq_{q_a}.

2.3 Maximum Sub-Optimality (MSO)

We now present the performance metrics to quantify the robustness of query processing. A traditional query optimizer will first estimate q_e, and then use P_{q_e} to execute a query which may actually be located at q_a. The sub-optimality of this plan choice, relative to an oracle that magically knows the correct location, and therefore uses the ideal plan P_{q_a} , is defined as:

$$SubOpt(q_e, q_a) = \frac{Cost(P_{q_e}, q_a)}{Cost(P_{q_a}, q_a)} \tag{2.1}$$

The quantity $SubOpt(q_e, q_a)$ ranges over $[1, \infty)$. With this characterization of a specific (q_e, q_a) combination, the maximum sub-optimality that can potentially arise over the entire ESS is given by

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

The above definition for a traditional optimizer can be generalized to selectivity discovery algorithms like `SpillBound`. Specifically, suppose the discovery algorithm is currently exploring a location $q \in Seq_{q_a}$ it will choose P_q as the plan and $Cost(P_q, q)$ as the associated budget. Extending this to the whole sequence, the analogue of Equation 2.1 is defined as follows:

$$SubOpt(Seq_{q_a}, q_a) = \frac{\sum_{q \in Seq_{q_a}} Cost(P_q, q)}{Cost(P_{q_a}, q_a)}$$

leading to

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

2.4 Assumptions

The primary assumptions made in this report that allow for systematic construction and exploration of the ESS are those of plan cost monotonicity (PCM) and predicate selectivity

independence (SI). PCM may be stated as: For any two locations $q_b, q_c \in ESS$, and for any plan P,

$$q_b \succ qc \Rightarrow Cost(P, q_b) > Cost(P, q_c)$$

That is, it encodes the intuitive notion that when more data is processed by a query, signified by the larger selectivities for the predicates, the cost of the query processing also increases.

On the other hand, SI assumes that the selectivities of the epps are all independent while this is a common assumption in much of the query optimization literature, it often does not hold in practice

While arbitrary selectivity estimation errors are permitted in our study, we have assumed the optimizer's cost model to be perfect that is, only optimizer costs are used in the evaluations, and not actual run times. While this assumption is certainly not valid in practice, improving the model quality is, in principle, an orthogonal problem to that of cardinality estimation accuracy.

2.5 Problem Definition

We denote the set of all the join and base filter predicates in the query as its *raw* error-prone predicate set. With the above framework, the problem of MSO-safe dimensionality reduction of the ESS is defined as follows:

For a given input query Q with its raw EPP set, develop a time-efficient pre-processing algorithm that identifies and removes maximum number dimensions (epps) from the ESS without worsening the original MSO guarantees.

Notation	**Meaning**
epp (EPP)	Error-prone predicate
ESS	Error-prone Selectivity Space
D	Dimensionality of the ESS
$e_1, ..., e_D$	The D epps in the query
$q \in [0,1]^D$	A location in the ESS space
$q.j$	selectivity of q in the j^{th} dimension of ESS
P_q	Optimal Plan at $q \in ESS$
q_a	Actual selectivity
$Cost(P, q)$	Cost of Plan P at location q

Table 2.1: Notations

Chapter 3

Overheads Reduction

3.1 Schematic Removal of Dimensions

Modern database systems maintain statistical information about the data stored in various relations in the form of various types of histograms for maintaining the number of unique values, the counts most frequently occurring elements among others. These histograms capture the data distribution and other characteristics of data which are useful in estimating the selectivity of a predicate with minimal effort. The selectivity estimates made using these meta data statistics may or may not be accurate and may lead to selectivity estimation errors and bad plan choices. But we can make use of this meta data and certain physical schema structures to deterministically guarantee the accuracy of estimates for certain epps.

We now present an approach for schematic removal of dimensions, this approach is particularly applicable for base predicates. It is based on the notion that if we know the selectivity of any predicate with a very high certainty then that dimension can safely be removed from the set of error prone predicates without incurring degradation in the query processing quality.

1. *Using Meta Data Statistics*

As stated above various meta data structures like histograms maintain statistical information about the data. Usually the exact frequency counts for most commonly occurring values in the column are stored explicitly in almost all databases. If the predicate queries the column on such a value whose exact frequency count exists apriori then the selectivity estimate for it can be made accurately. We have done explicit verification of these facts on the open source *PostgreSQL* [7] and the commercial *SQL Server* [6] engine.

In case of range predicates we can predict its selectivity accurately only if the bucket boundaries of the histogram exactly coincide with the range specified in the query.

Nevertheless in case of equality predicate if the frequency of the queried value is not maintained explicitly then we can assume its lower bound selectivity to be 0 and the upper-bound selectivity to be the size of the bucket in which its value happens to fall.

Similar argument applies to the range predicates whose range does not coincide with the bucket boundaries exactly. The lower-bound selectivity is then the summation of the bucket frequencies which entirely fall within the given range and the upper-bound is the lower-bound selectivity plus the bucket frequencies of the buckets which partially fall in the specified range. These bounds are deterministic although may be crude, but are of great value as we will see in Section 3.2.5.

For using meta data statistics for Schematic removal of dimensions it is important that the statistics must be up-to-date.

2. *Using Physical Schema*

Let us consider the example base predicate *cd_education_status='Unknown'* from TPC-DS Query 27. If an index already exists on the column *cd_education_status* then accessing the index structure provides accurate selectivity estimate which aids in making the optimal plan choice. Such dimensions can be readily removed from the set of *epps*.

Accessing indexes for accurately estimating selectivity can be costly but may be worth depending on the database execution setting. For instance, in OLAP settings with rich physical schema environments where heavy weight queries (typically takes hours to run) are expected to be executed, using the index structures for not only accessing the relations but for making accurate selectivity estimations can reap huge benefits.

Further equality predicates on large domain columns queried for an uncommon value can be hard to estimate using typical meta data structures, using indexes provides a safety net from making heavy errors in selectivity estimates. Also, unlike the meta-data structures, indexes are always up-to-date.

3.2 Dimension Removal via Projection

Once the schematic removal of dimensions is done, we know that for the remaining dimensions optimizer has no way to estimate their selectivities with high accuracy. In the Projection technique we project the candidate dimension(s) to be removed to its (their) maximum possible selectivity value (or its upper-bound as calculated in Section 3.1). Hence by the virtue of PCM we always have sufficient cost budget to process the query irrespective of the actual selectivity values of the projected dimensions. But forcing a dimension to its maximum value may cause

sub-optimal plan choices since the actual value could be different from the projected value. To measure the impact of this sub-optimality we calculate the factor by which the MSO gets inflated due to these sub-optimal plan choices, hereafter called as the `InflationFactor`. We now describe the technique in detail.

3.2.1 2-D Projection Algorithm

Figure 3.1 shows an example ESS with two dimensions X and Y respectively. We will describe the algorithm by removing dimension X from the example ESS. The algorithm constructs a 1D ESS which is essentially the projection of the 2D ESS on $sel(X) = 1$ line i.e. for dimension X we simply assume the highest selectivity which is 1. In the subsequent phase, we deploy SpillBound algorithm over the projected 1D ESS.

Before we start the analysis of the impact of the above proposed algorithm on MSO, first let us calculate the sub-optimality for an arbitrary location $q_a(x, y) \in$ 2D ESS. Further, let $q_a^{max} = (1, q_a.y)$ and $q_a^{min} = (0, q_a.y)$ represent the projections of the location $q_a \in$ 2D ESS on $sel(X) = 1$ and $sel(X) = 0$ line, respectively. For instance, the above mentioned projections for location q_a can be seen in the Figure 3.1.

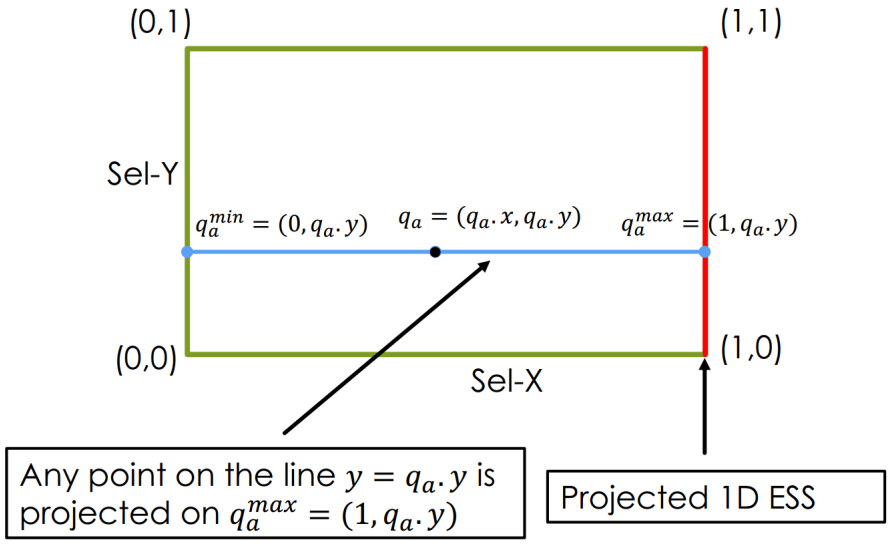


Figure 3.1: Example 2D ESS

Lemma 3.1 *The sub-optimality for any location $q_a \in$ 2D ESS is at most $4 * \frac{Cost(q_a^{max})}{Cost(q_a^{min})}$ after removing dimension X.*

Proof: Let the total cost incurred by the SpillBound algorithm for executing q_a^{max} be denoted

by $Cost_{SpillBound}(*, q_a^{max})$. Then the sub-optimality for q_a is

$$= \frac{Cost_{SpillBound}(*, q_a^{max})}{Cost(q_a)}$$

$$= \frac{Cost_{SpillBound}(*, q_a^{max})}{Cost(q_a^{max})} * \frac{Cost(q_a^{max})}{Cost(q_a)}$$

$$\leq 4 * \frac{Cost(q_a^{max})}{Cost(q_a)}$$

(MSO for 1D `SpillBound` [1] is 4.)

$$\leq 4 * \frac{Cost(q_a^{max})}{Cost(q_a^{min})}$$

□ The quantity $\frac{Cost(q_a^{max})}{Cost(q_a^{min})}$ which denotes the sub-optimality is called the `InflationFactor`. To analyze the impact of removal of X on MSO, we define the quantity `MaxInflationFactor` as

$$\texttt{MaxInflationFactor}(X) = \max_{q.y \in [0,1]} \frac{Cost(q^{max})}{Cost(q^{min})}$$

Corollary 3.1 *The MSO by removing a single dimension X (without loss of generality) from a 2D-ESS is at most* $(4 * \texttt{MaxInflationFactor}(X))$.

Now if $(4 * \texttt{MaxInflationFactor}) < 10$ (MSO `SpillBound` for 2D queries) then we can remove the dimension from the ESS, by assuming its selectivity to be 1, without any degradation in the MSO given by `SpillBound`.

Let us now analyze the computational effort required in calculating the `MaxInflationFactor` for the 2D scenario. The `MIF` for removing a dimension, say X, can be calculated without having to construct the complete 2D ESS, we just need to make optimizer calls for the two projected 1D segments where $X = 1$ and $X = 0$. Hence this requires just $2 * resolution$ optimization calls per candidate dimension removal.

3.2.2 Extending to higher Dimensions

In this sub-section, we show how to calculate the `MIF` for removing k dimensions from a D dimensional ESS where $D > 2$ using the Projection algorithm.

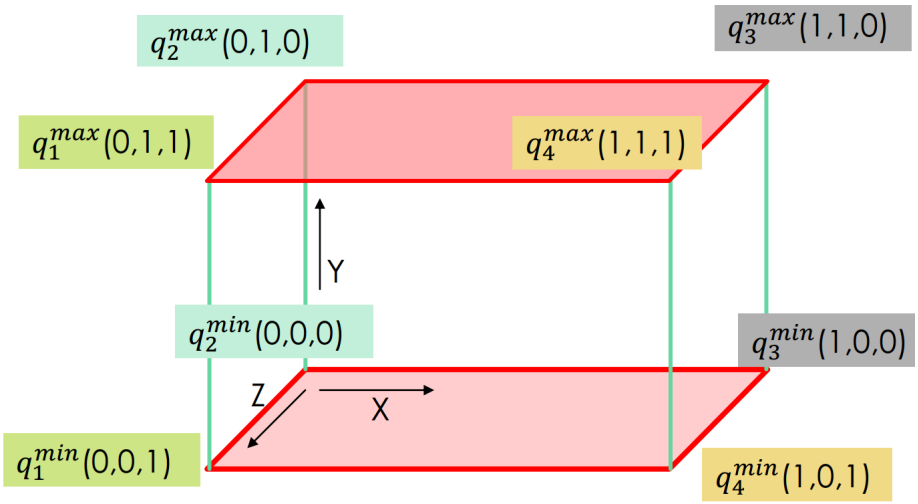


Figure 3.2: 3D ESS - `MaxInflationFactor`

The above 2-D algorithm can be extended to a multi-dimensional algorithm as follows. For a D-dimensional query to remove k dimensions, say $e_1, .., e_k$, we first construct two (D-k)-dimensional ESS sub-spaces by setting the selectivity of the dimensions $e_1, .., e_k$ to 0 and 1 respectively. We then compute the `InflationFactor` for every selectivity combination of the retained dimensions $e_{k+1}, .., e_D$. This is shown in Figure 3.2, where dimension Y is to be removed. The red colored 2-D slices correspond to the ESS subspaces where the selectivity of dimension $Y = 1$ *and* $Y = 0$. The ratio of the costs of each of the corresponding points in these slices are the `InflationFactors`. We now define,

$$\texttt{MaxInflationFactor}_k(e_1, .., e_k) = \max_{\forall(q.j_{k+1},..,q.j_D)\in[0,1]^{D-k}} \frac{Cost(q^{max})}{Cost(q^{min})} \tag{3.1}$$

where q^{max} corresponds to the location where $q.i = 1$ and q^{min} corresponds to the location where $q.i = 0$ where $i \in \{1, .., k\}$.

Removal of dimensions using the above mentioned algorithm has advantage if

$$\texttt{MaxInflationFactor}_k(e_1, .., e_k) * MSO_{SB}(D-k) < MSO_{SB}(D)$$

Order of Dimension Removal : Choice and number of the dimensions to be removed plays an important role in deciding the degree to which the dimensionality of the query can be reduced by minimally affecting the MSO guarantees. We first calculate the `MIF` for each of the dimensions individually, then the sequence for removing multiple dimensions is decided greedily in the increasing order of the `MIFs` corresponding to the individual dimensions. We have verified

extensively enumerating all the possible orders for removal that this greedy order of removal often results in the optimal order and in the other cases it is near optimal.

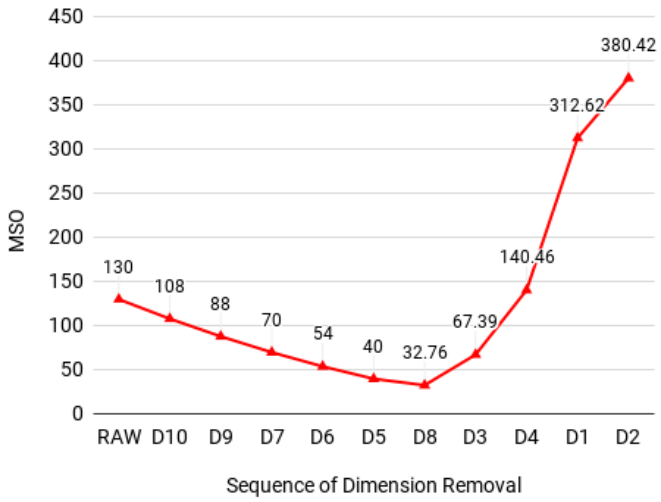


Figure 3.3: MSO graph for Multiple Dimension Removal

Figure 3.3 shows the MSO for sequential sequential removal of dimensions for the TPC-DS query 27. *RAW* denotes the MSO corresponding to the initial ten dimensions, then we remove dimensions in a sequential fashion and calculate the MSO after each removal. The MSO graph is similar to a cup shape where MSO initially declines as we remove the dimensions and after a certain point (the base of the cup), it begins to increase. In Figure 3.3 the minimum MSO of 32.76 is attained when dimensions {*D10,D9,D7,D6,D5,D8*} are removed with an `MIF` of 1.17 as presented earlier. We make the choice to stop the dimension removal at *D8* since removing any dimension after it worsens the previous MSO.

Computational Effort : The computation efforts, in terms of optimization calls, needed for calculating the `MIF` grows exponentially with increasing number of dimensions. In the above case for constructing 2D number of (D-1) dimensional ESS sub-spaces for calculating the individual `MIFs` (projections of the dimensions to be removed) we require precisely $\theta(D*res^{D-1})$ optimization calls. Then for removing k dimensions sequentially we require $\theta(2*res^{D-k})$ optimization calls per iteration where $k = 1 \rightarrow (D-1)$ which is $O(res^{D-1})$ calls which is very expensive. We will now see how to mitigate this problem in the next subsection.

3.2.3 `MaxInflationFactor` Perimeter

To obtain a reduction in the computational efforts for removing the dimensions we need to devise a strategy to calculate the `MIF` deterministically in a computationally cost effective manner.

We now present a computationally cheap algorithm `MaxInflationFactor` perimeter which computes the `MIF` by just making optimization calls along the *perimeter* of the ESS (1-D simplexes of the ESS) under certain conditions.

The number of optimization calls for calculating `MIF` at perimeter of the ESS is of the order $\theta(2^{D-1} * D * resolution)$. The lemmas for perimeter `MIF` are based on the replacement safety conditions in [4].

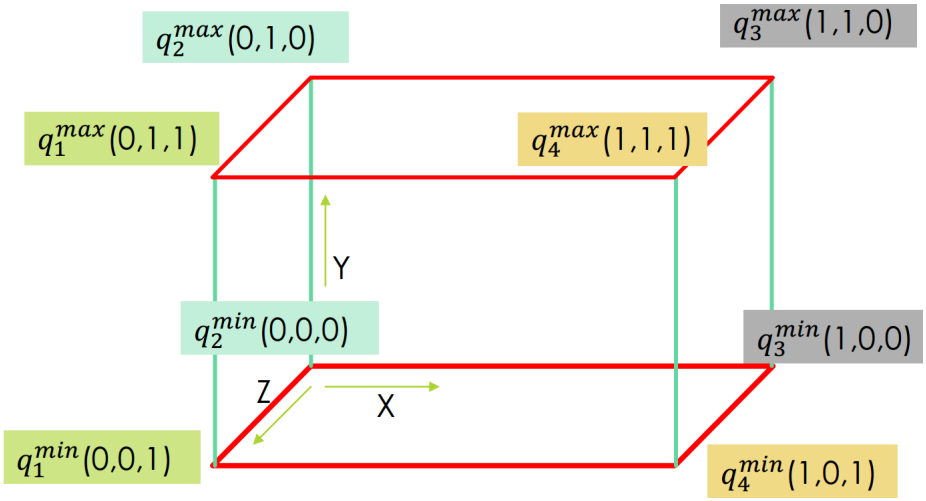


Figure 3.4: 3D ESS - `MaxInflationFactor` Perimeter

Let us consider the 3-D ESS as shown in Figure 3.4 where we want to calculate the `MaxInflationFactor` for removing dimension Y. The thick-red lines denote the perimeter for the slice where $sel(Y) = 1$ and for the slice where $sel(Y) = 0$. We now investigate if making optimization calls only at the perimeter are sufficient to correctly calculate the `MaxInflationFactor`. Let m denote `InflationFactor` function.

$$m(x, z) = \frac{Cost(q.x, q.y = 1, q.z)}{Cost(q.x, q.y = 0, q.z)}$$

The optimal cost surface over the ESS is called the Plan Infimum Cost Curve(PIC). Here we make an empirically verified assumption that the PIC is an axis parallel monotonically increasing piecewise linear function in X, Y and Z. Let f denote the axis parallel linear PIC function (this is a stronger assumption but we show later how we can relax it for piecewise axis parallel linear functions as well). Let us call this function as *3D APL* function and let it have the form

$$f(x, y, z) = k_1x + k_2y + k_3z + k_4xz + k_5yz + k_6xz + k_7xyz + k_8 \tag{3.2}$$

When dimension $Y = 1$ we can substitute its value and the PIC can be represented as

$$f|_{Y=1} = a_1x + a_2z + a_3xz + a_4, \tag{3.3}$$

analogously when $Y = 0$ the PIC is

$$f|_{Y=0} = b_1x + b_2z + b_3xz + b_4 \tag{3.4}$$

The `InflationFactor` function m is now a division of two *2D ALP* functions $f|_{Y=1}$ and $f|_{Y=0}$ i.e.

$$m(x, z) = \frac{f|_{Y=1}}{f|_{Y=0}} = \frac{a_1x + a_2z + a_3xz + a_4}{b_1x + b_2z + b_3xz + b_4} \tag{3.5}$$

The function $m_z(x)$ denotes the sub-optimality ratios by keeping Z constant and varying along dimension X (and the vice-versa for $m_x(z)$). The various possible behaviors of $m_{z_o}(x)$ are shown in Figure 3.5 as Curves (a) through (d). This behavior can be attributed to the division of two *2D APL* functions in Equation 3.5.

Now the reduced ESS after removing Y has two dimensions X and Z. Let the reduced ESS be denoted by $ESS_R := \{(x_1, z_1), (x_1, z_2), (x_2, z_1), (x_2, z_2)\}$, where $x_1 = 0, z_1 = 0, x_2 = 1$ *and* $z_2 = 1$. The maximum `InflationFactor` value in any subspace of the ESS_R is called the *local* `MaxInflationFactor`.

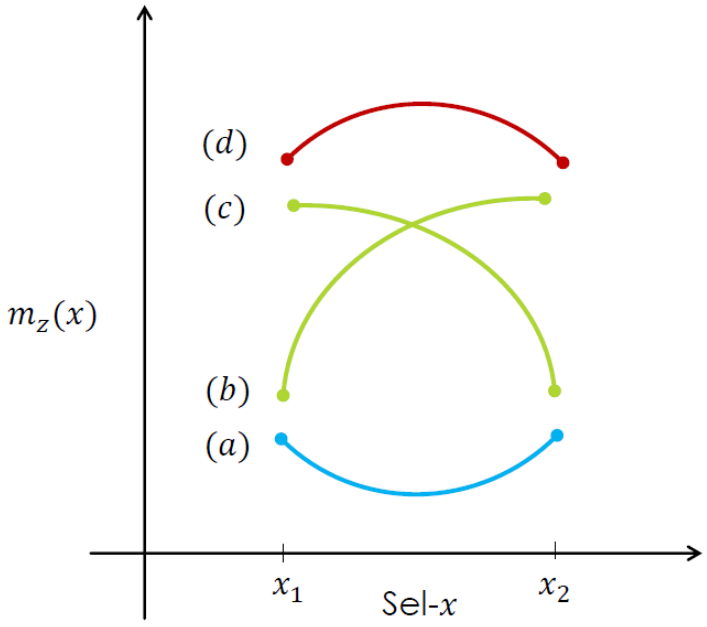


Figure 3.5: Behavior of $m_z(x)$ function

Lemma 3.2 *Given a fixed $Z = z_o$ and the pair of points (x_1, z_o) and (x_2, z_o), the local* `MaxInflationFacto` *occurs at one of these points if the slope $m'_{z_o}(x)$ is either*

1. *monotonically non-decreasing, OR*
2. *monotonically decreasing with $m'_{z_o}(x_1) \leq 0$ or $m'_{z_o}(x_2) \geq 0$*

A similar result holds when x is fixed.

Proof: When the slope of $m'_{z_o}(x)$ is monotonically non-decreasing (i.e. Condition (1) is satisfied), the `InflationFactor` function curve that connects the two points is guaranteed to lie below the straight line joining the two points Curve (a) in Figure 3.5 shows an example of this situation. This ensures that the `InflationFactor` along the given line segment is always less than or equal to the `InflationFactor` at one of the end-points of the segment.

If, on the other hand, $m'_{z_o}(x)$ is monotonically decreasing, then the possible behaviors of the `InflationFactor` function $m_{z_o}(x)$) are shown in curves (b) through (d) in Figure 3.5. Curves (b) and (c) denote the behavior of the `InflationFactor` function when Condition (2) is satisfied, and clearly the value of the `InflationFactor` function is below at least one end-point in the given range. Curve (d) however represents scenario where the local `MaxInflationFactor` does not occur at one of the end-points and hence does not satisfy either conditions of the lemma. □

Lemma 3.3 *If the slope of the* `InflationFactor` *function $m'_z(x)$, is non-decreasing (resp. decreasing) along the line-segments $z = z_1$ and $z = z_2$, then it is non-decreasing (resp.decreasing) for all line segments in the interval (z_1, z_2). A similar result holds for $m'_x(z)$.*

Proof: The `InflationFactor` function given in Equation 3.5 for fixed z is

$$m_z(x) = \frac{c_1 x + c_2}{d_1 x + d_2}$$

where

$$c_1 = (a_1 + a_3 z),\ c_2 = (a_2 z + a_4),\ d_1 = (b_1 + b_3 z) \text{ and } d_2 = (b_2 z + b_4) \tag{3.6}$$

Consider the slope of this `InflationFactor` function

$$m'_z(x) = \frac{c_1 d_2 - c_2 d_1}{(d_1 x + d_2)^2} \tag{3.7}$$

For $x \in (0, 1]$, this slope is monotonic and its behavior depends on the sign of the numerator $N := c_1 d_2 - c_2 d_1$. From Equations 3.3, 3.4 and 3.6 we know that the numerator of the slope N can be written as the following function of z

$$N(z) = (a_1 + a_3 z)(b_2 z + b_4) - (a_2 z + a_4)(b_1 + b_3 z)$$

$$= (a_3 b_2 - a_2 b_3) z^2 + (a_1 b_2 + a_3 b_4 - b_1 a_2 - a_4 b_3) z + a_1 b_4 - b_1 a_4$$

$$= l_1 z^2 + l_2 z + l_3 \tag{3.8}$$

where l_1,l_2 and l_3 are constants. Now since $N(z)$ is a quadratic function of z, if $N(z_1) \geq 0$ *and* $N(z_2) \geq 0$ then $N(z) \geq 0$, $\forall z \in (z_1, z_2)$ which implies that slope $m'_z(x)$ is decreasing $\forall z \in (z_1, z_2)$. Hence the Lemma immediately follows. □

Theorem 3.1 *(Perimeter* `MaxInflationFactor`*). For the reduced selectivity space ESS_R with corners $[(x_1, z_1), (x_1, z_2), (x_2, z_1), (x_2, z_2)]$, calculating the* `InflationFactor` *along the perimeter is sufficient for finding the* `MaxInflationFactor` *if any one of the conditions C1 through C6, given in Table 3.1 is satisfied.*

Proof:

Consider the C1 condition in Table 3.1: Since $m''_z(x) \geq 0$ (i.e slope $m'_z(x)$ is non-decreasing) at the T-B boundaries, then from Lemma 3.3, we know that the slope $m'_z(x)$ is non-decreasing throughout the range (z_1, z_2).

Moving on to the C2 and C3 conditions: Since $m''_z(x) < 0$ (i.e slope $m'_z(x)$ is decreasing) at the T-B boundaries, then from Lemma 3.3, we know that the slope $m'_z(x)$ is decreasing throughout the range (z_1, z_2). Further, we know that for a given $z = z_o \in (z_1, z_2)$, either $m'_{z_o}(x_1) \leq 0$ (C2) or $m'_{z_o}(x_2) \geq 0$ (C3).

Thus, when C1, C2 or C3 is satisfied, then for all lines between points (x_1, z) and (x_2, z), $z \in (z_1, z_2)$, the local `MaxInflationFactor` occurs at one of the end-points of these lines as a result of the slope conditions given in Lemma 3.2 are satisfied. Since the union of all such line-segments is the given region, `MaxInflationFactor` occurs at the perimeter. Similar arguments can be used to show that the perimeter is sufficient for finding the `MaxInflationFactor` when conditions C4, C5 or C6 are satisfied.

Table 3.1: Perimeter `MaxInflationFactor` Conditions

	Left Boundary	**Right Boundary**	**Top Boundary**	**Bottom Boundary**
C1	-	-	$m''_{z_2}(x) \geq 0$	$m''_{z_1}(x) \geq 0$
C2	$m'_z(x_1) \leq 0$	-	$m''_{z_2}(x) < 0$	$m''_{z_1}(x) < 0$
C3	-	$m'_z(x_2) \geq 0$	$m''_{z_2}(x) < 0$	$m''_{z_1}(x) < 0$
C4	$m''_{x_1}(z) \geq 0$	$m''_{x_2}(z) \geq 0$	-	-
C5	$m''_{x_1}(z) < 0$	$m''_{x_2}(z) < 0$	$m'_x(z_2) \geq 0$	-
C6	$m''_{x_1}(z) < 0$	$m''_{x_2}(z) < 0$	-	$m'_x(z_1) \leq 0$

□

For extension to higher dimensions we need to consider the 2-D simplexes (2D-faces) of the higher dimensional ESS. If all the 2D-simplexes of the higher dimensional ESS satisfy any of the conditions from C1 to C6 in Table 3.1 then the interior becomes safe and `MaxInflationFactor` occurs at the perimeter by Theorem 3.1.

3.2.4 `MaxInflationFactor` Perimeter Extension to Axis Parallel Piecewise Linear Functions

In the real-world scenario the PIC is usually not the *APL* function described in Equation 3.2, but by observing the PIC curve we have empirical evidence that the PIC curves exhibit axis-parallel piece-wise linearity [3].

The idea now is to divide the domain of the PIC into non-overlapping regions such that in each region the PIC is an *APL* function. Now if the perimeter of each of these regions satisfy any of the conditions from C1 to C6 of Table 3.1 then by Theorem 3.1 the `MaxInflationFactor` occurs on the perimeter.

To identify these regions we explicitly fit the PIC on the perimeter with piecewise linear functions. The breakpoints of these linear pieces on the perimeter are then used to divide the domain of the PIC into non-overlapping regions. The perimeter test can then be used on these regions in isolation.

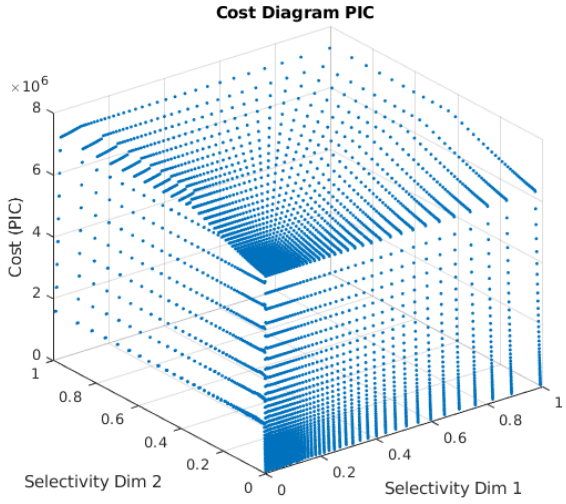
Now the question remains whether this type of fit is good enough, our experiments show that this methodology gives us good fits with low RMSE values. The important point to note here is that we do not require an exact fit with accurate co-efficients but we need to capture the slope behavior of the PIC.

Let us consider an example PIC of the TPC-DS query 26, generated using repeated invocations of the *PostgreSQL* optimizer. This is shown in Figure 3.6a. The 2D input domain of the PIC, which is the 2D selectivity region spanned by dimensions 1 and 2 is divided into 9 regions.

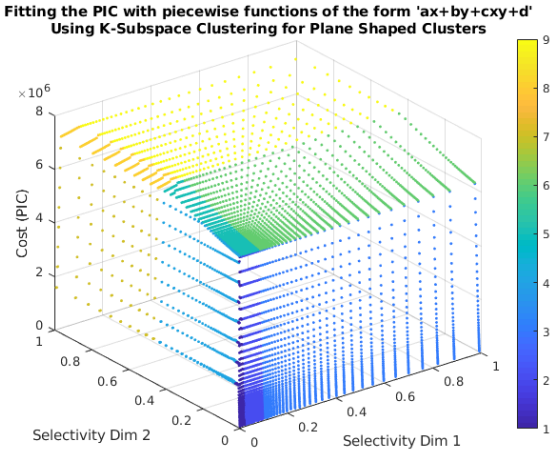
Each region is then fitted with the 2D APL function of the form,

$$f(x, y) = ax + by + cxy + d \tag{3.9}$$

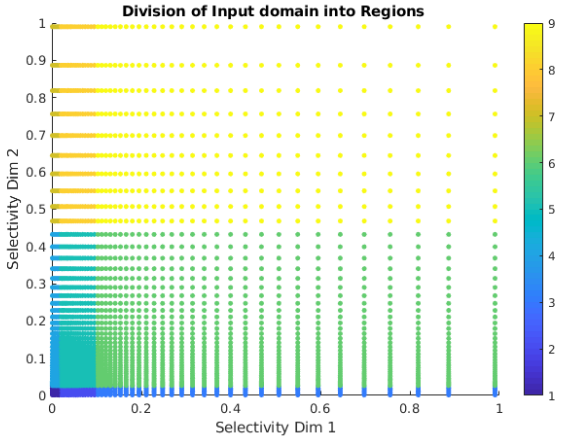
We use non-linear least squares regression to fit the function and we are able to do so with normalized RMSE = 9 %. The projection of the boundaries of these regions on the input domain is shown in Figure 3.6c. The computational complexity of the `MIF` perimeter algorithm for k regions is $O(2^{D-1} * D * res * k)$.



(a) PIC



(b) PIC fitted with piecewise functions



(c) Dividing Input Domain by projecting the Clusters

Figure 3.6: Original PIC and PIC fitted with piecewise functions as Equation 3.9

The problem now is identification of these regions where the Function 3.9 fits nicely. This is done using the K-subspace clustering methods for 2D and higher dimensional planes as described in [5] shown in Figure 3.6b. But using k-subspace clustering methods for identifying and fitting the planes we need to provide the PIC values as an input to the algorithm. This implies explicitly discovering the ESS which is a very expensive approach.

Hence we try to identify these regions using only PIC at perimeter. The perimeter constitutes of 1D simplexes, the PIC at each of the 1D simplexes is a piecewise linear 1D function. For each of these 1D PICs we do the following, we first fit the 1D PIC with a piecewise linear function. This operation is done by using the K-subspace clustering [5] method for line shaped clusters shown in Figures 3.7a and 3.7b. The basic idea is as follows,

1. They represent a line by a point in space and unit direction. If the point is c_k and the direction is a_k then a data point x can be decomposed into a parallel component $x^{\parallel} = a_k[a_k^T(x - c_k)]$ and a perpendicular component $x^{\perp} = (x - c_k) - x^{\parallel}$.

2. The distance between the point x and cluster C_k is defined as the perpendicular distance between the point and the line :

$$Dist(x, C_k) = \parallel x - c_k - \alpha a_k \parallel^2$$

   where $\alpha = (x - c_k)^T a_k$.

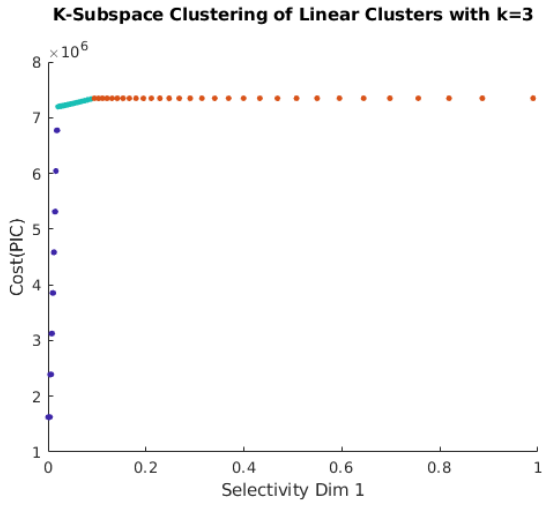
3. The objective function is to minimize the total distance (dispersion) of the cluster, which is $\min_{c_k, a_k} \sum_{i \in C_k} Dist(x_i, C_k)$.

4. They then give a Theorem that using the model parameters $c = \bar{x}^{(k)}$ where $\bar{x}^{(k)} = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$ is the centroid of the cluster and $a = u_1$ where u_1 is the first principal direction obtained by the principal component analysis(PCA) of the cluster points, we obtain the optimal solution of the objective function stated previously.

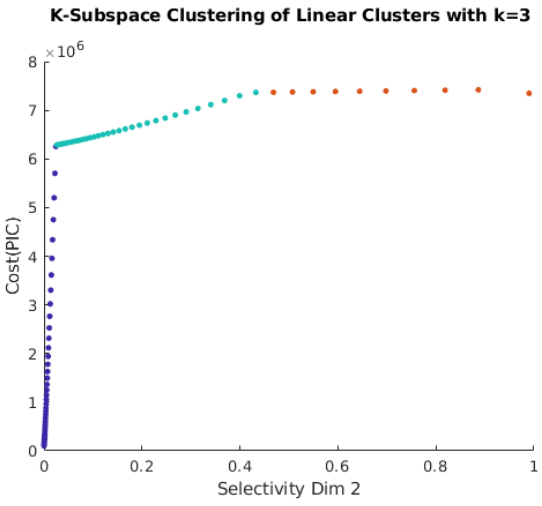
We then identify the breakpoints of these linear pieces. We then use the combination of the breakpoints from the different 1D simplexes to form 2D regions in the input domain. This is shown in Figure 3.7c which is almost identical to the domain decomposition in Figure 3.6c which was obtained using the PIC values for the entire region.

We then experimentally verify if the Function 3.9 fits well to the PIC restricted to the regions obtained from the combinations of the breakpoints of the piecewise linear functions at perimeter. The fit is done for PICs of 8 TPCDS queries with an average normalized RMSE of 12%.

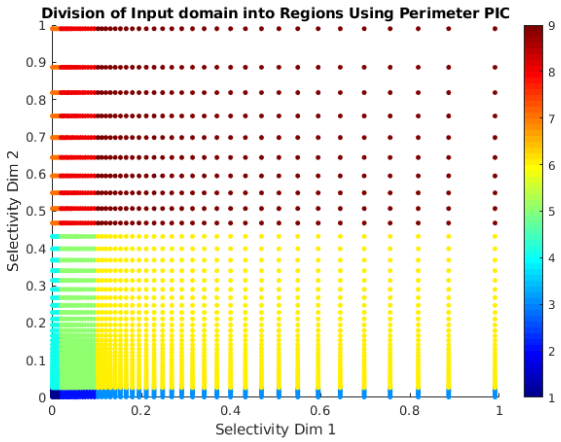
If the `InflationFactor` function (m) satisfies anyone the conditions in Table 3.1 then the `MaxInflationFactor` occurs at the perimeter of one of these regions, since they together constitute the entire input domain.



(a) 1D PIC at perimeter Side 1



(b) 1D PIC at perimeter Side 2



(c) Dividing Input Domain using PICs at perimeter

Figure 3.7: K-Subspace Clustering of 1D PICs with linear clusters

The calculation of `MIF` on perimeter using the above algorithm is shown in Table 3.2. It also compares the `MIF` values obtained using perimeter with brute force calculation of `MIF` done by generating the entire ESS. The perimeter `MIF` technique does exceptionally well as it is able to calculate the exact value of `MIF` using only the perimeter of the ESS for all the queries except Q-84. On investigation it is observed that the PIC for Q-84 violates even basic assumptions like PCM let alone *APL*.

Table 3.2: `MaxInflationFactor` Calculation using BruteForce, Vertices and Perimeter

QT	Technique	Dim 1	Dim 2	Dim 3	Dim 4	Dim 5	Dim 6
DSQT264D	MIF-BruteForce	3.843	135.539	10.350	192.936		
	MIF-Vertices	3.843	135.539	10.350	192.936		
	MIF-Perimeter	3.843	135.539	10.350	192.936		
DSQT74D	MIF-BruteForce	2.969	10.152	15.729	228.866		
	MIF-Vertices	2.969	10.152	15.729	228.866		
	MIF-Perimeter	2.969	10.152	15.729	228.866		
DSQT153D	MIF-BruteForce	49.274	57.083	353.012			
	MIF-Vertices	7.226	56.546	346.205			
	MIF-Perimeter	49.274	57.083	353.012			
DSQT195D	MIF-BruteForce	9.853	133.528	70.551	100.555	90.239	
	MIF-Vertices	9.853	115.082	57.477	100.555	90.239	
	MIF-Perimeter	9.853	133.528	57.477	100.555	90.239	
DSQT274D	MIF-BruteForce	10.220	15.883	159.238	2.978		
	MIF-Vertices	10.220	15.883	159.238	2.978		
	MIF-Perimeter	10.220	15.883	159.238	2.978		
DSQT295D	MIF-BruteForce	14.210	165.084	166.317	227.236	21.813	
	MIF-Vertices	13.775	165.084	166.317	227.236	21.079	
	MIF-Perimeter	14.210	165.084	166.317	227.236	21.813	
DSQT845D	MIF-BruteForce	99.159	98.239	2467.991	9180.578	53.630	
	MIF-Vertices	37.240	37.112	76.369	1679.043	39.360	
	MIF-Perimeter	96.422	95.552	1895.860	3404.853	53.630	
DSQT912D	MIF-BruteForce	30.412	1.920				
	MIF-Vertices	30.412	1.920				
	MIF-Perimeter	30.412	1.920				
DSQT913D	MIF-BruteForce	492.859	5.313	69.838			
	MIF-Vertices	492.859	5.313	69.838			
	MIF-Perimeter	492.859	5.313	69.838			
DSQT914D	MIF-BruteForce	273.807	13.581	11.146	169.250		
	MIF-Vertices	273.807	13.581	11.146	169.250		
	MIF-Perimeter	273.807	13.581	11.146	169.250		
DSQT915D	MIF-BruteForce	273.203	3.156	13.542	11.115	168.830	
	MIF-Vertices	273.203	2.233	13.542	11.115	168.830	
	MIF-Perimeter	273.203	3.156	13.542	11.115	168.830	
DSQT963D	MIF-BruteForce	952.858	189.084	891.017			
	MIF-Vertices	952.858	189.084	891.017			
	MIF-Perimeter	952.858	189.084	891.017			
DSQT186D	MIF-BruteForce	83.037	9.167	3.054	81.633	67.287	95.696
	MIF-Vertices	64.609	9.167	3.054	35.160	39.336	58.962
	MIF-Perimeter	83.037	9.167	3.054	81.633	67.287	95.696
DSQT916D	MIF-BruteForce	471.437	272.993	3.155	13.517	11.094	168.523
	MIF-Vertices	471.437	272.993	2.229	13.517	11.094	168.523
	MIF-Perimeter	471.437	272.993	2.229	13.517	11.094	168.523

3.2.5 Using Selectivity Estimate bounds to minimize `MaxInflationFactor`

For some dimensions we can obtain deterministic lower and upper bounds on the selectivity using the approaches presented in Section 3.1. The ESS for such dimensions can be shrunk leading

to smaller values of the `MaxInflationFactor`. The perimeter `MIF` algorithm described above is directly applicable to this ESS with shrunken dimensions. In general the `MaxInflationFactor` for removing k dimensions is given by Equation 3.1.

Now let us define q_{ub}^{max} where $q.i = ub(sel_i)$ and q_{lb}^{min} where $q.i = lb(sel_i)$ where $i \in DimRem$. Also let $(q_{j_1}, ..., q_{j_{D-k}}) \in (lb(sel_{j_1}), ub(sel_{j_1})) \times ... \times (lb(sel_{j_{D-k}}), ub(sel_{j_{D-k}}))$. Now since $q^{max} \succ q_{ub}^{max}$ and $q_{lb}^{min} \succ q^{min}$, therefore by PCM for any q

$$\frac{Cost(q^{max})}{Cost(q^{min})} \geq \frac{Cost(q_{ub}^{max})}{Cost(q_{lb}^{min})}$$

Hence using Selectivity Estimate Bounds helps to lower the `MaxInflationFactor`. The effect is significant as these bounds get narrower and tighter.

Chapter 4

Performance Improvement

4.1 Contour Plan Replacement

After the removal of k dimensions using Projection and Schematic techniques, we construct the ESS from the remaining dimensions incurring a computational effort of (res^{D-k}) optimization calls. Once the ESS is constructed and we have discovered the iso-cost contours we exploit the weakness of a dimension by using contour plan replacement technique. The order of the dimensions chosen, against which plan replacement is to be done, is decided by their `MaxInflationFactors`, which in essence quantifies the weakness of a dimension based on the impact of its selectivity to the optimal cost surface.

The MSO of the `SpillBound` algorithm is the result of at most D fresh executions per contour and at most $\frac{D(D-1)}{2}$ repeat executions overall as stated in [1]. To reduce the MSO for a query we explore the possibility of guaranteeing less number of fresh executions per contour and lower number of repeat executions overall.

Now the question is how do we reduce the number of executions per contour, the idea is *piggybacking* the execution of "weak" dimensions along the execution of "strong" dimensions. By execution of a dimension, we mean execution of the corresponding predicate. This *piggybacking* of executions now inflates the budget required for the execution of strong dimensions. The objective is to make the MSO a function of the strong dimensions modulo the inflation of cost budgets for the *piggybacked* execution of weak dimensions along them. These inflations in budgets are expected to be low for dimensions having low `MIF` values because variation in the selectivities of these weak dimensions have less impact on the cost of the overall query.

In the following sub-sections we show how to achieve the piggybacked executions and how to deterministically calculate the inflation in budgets.

4.1.1 Contour Plan Replacement along single dimension

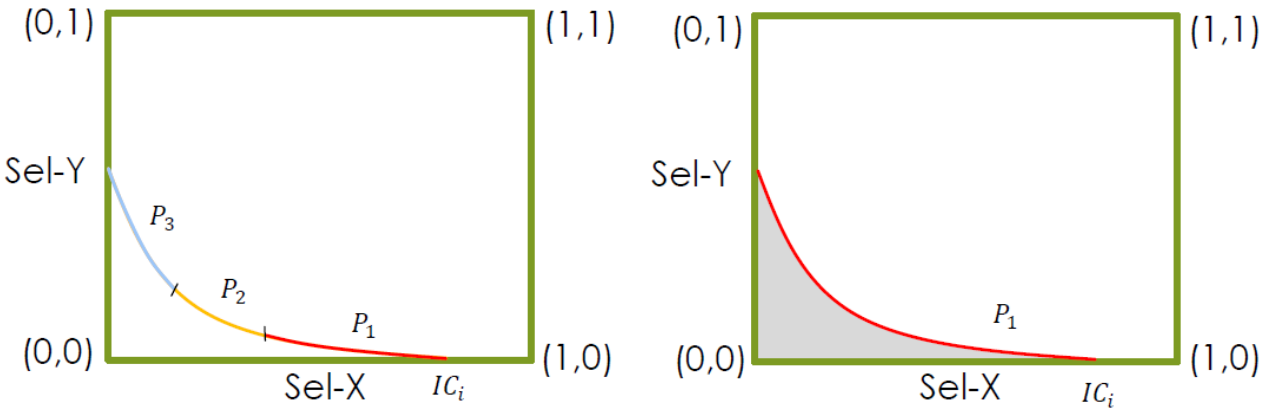
We will first describe our algorithm for plan replacement along a single dimension say e_D. For a D dimensional query the goal is to bound the maximum number of fresh executions per contour to $(D-1)$ and maximum repeat executions to $\frac{(D-1)(D-2)}{2}$ so as to achieve the MSO of a $(D-1)$ dimensional query by incurring a small penalty λ^{e_D} such that $\lambda^{e_D} * MSO_{D-1} < MSO(D)$. The extension for contour plan replacement along multiple dimensions is straightforward.

4.1.1.1 2-D Scenario

Consider the 2-D ESS as shown in Figure 4.1a,which depicts the iso-cost contour IC_i annotated with the optimal plans P_1, P_2 and P_3 covering it. We now describe the 2-D algorithm for plan replacement along dimension X.

For the contour IC_i we find the best one-plan replacement (from the contour POSP set) which can cover the entire contour i.e replace all the plans across it with minimum sub-optimality (λ_i^X). For instance as shown in the Figure 4.1b, plan P_1 replaces all other plans on the contour IC_i with sub-optimality (λ_i^X).

In the scenario that X-dimension needs to be removed, all plans that are X-spilling are considered to be Y-spilling by ignoring the error-prone X-predicate in the pipeline order.



(a) Contour on a 2-D ESS

(b) Best 1-plan replacement along the contour

Figure 4.1: Contour Plan Replacement 2D Scenario

Lemma 4.1 *Consider the contour plan P_r which replaces all the plans on contour IC_i with sub-optimality λ_i^X. Let epp Y be selected by the spill node identification mechanism (after ignoring the epp X). When P_r is executed with budget $CC_i(1+\lambda_i^X)$ and spilling on Y, then we either learn (a) the exact selectivity of Y, or (b) that q_a lies beyond the contour.*

Proof: IC_i represents the set of points in the ESS having their optimal cost equal to CC_i.

Let q^x_{sup}, q^x_{inf}, q^y_{sup} and q^y_{inf} denote the points having maximum and minimum X-selectivity and Y-selectivity on the contour respectively. The cost of all points $q \in IC_i$ is at most $CC_i(1+\lambda^X_i)$ when costed using P_r. Now when the plan P_r is executed in the spill-mode with cost budget $CC_i(1+\lambda^X_i)$ it may or may not complete.

For an internal node N of a plan tree, we use $N.cost$ to refer to the execution cost of the node. Let N_Y denote the internal node corresponding to Y in plan P_r. Partition the internal nodes of P_r into the following: $Upstream(N_Y)$, $\{N_Y\}$, and $Residual(N_Y)$, where $Upstream(N_Y)$ denotes the set of internal nodes of P_r that appear before node N_Y in the execution order, while Residual (N_Y) contains all the nodes in the plan tree excluding $Upstream(N_Y)$ and $\{N_Y\}$. Therefore, $Cost(P_r, q) = \sum_{N \in Upstream(N_Y)} N.cost + N_Y.cost + \sum_{N \in Residual(N_Y)} N.cost$

Case-1 : The value of the first term in the summation $Upstream(N_Y)$ is known with certainty if it does not contain N_X. Further, the quantity $N_Y.cost$ is computed assuming that the selectivity of N_Y is $q.y$ for any point $q \in IC_i$. Since the output of N_Y is discarded and not passed to downstream nodes, the nodes in $Residual(N_Y)$ incur zero cost. Thus, when P_r is executed in spill-mode, the budget is sufficiently large to either learn the exact selectivity of Y (if the spill-mode execution goes to completion) or to conclude that $q_a.y$ is greater than $q.y$, $\forall q \in IC_i$ since P_r is costed for all $q \in IC_i$.

Case-2 : Now if N_X is contained in $Upstream(N_Y)$ then its cost is not known with certainty, however since P_r is costed for all $q \in IC_i$, all the selectivity combinations of $(q.x, q.y)$, $\forall q \in IC_i$ get considered. Hence, for all these combinations the sum of the quantity $\sum_{N \in Upstream(N_Y)} N.cost +$ $N_Y.cost \leq CC_i(1+\lambda^X_i)$. Similar to Case-1, the output of N_Y is discarded and not passed to downstream nodes, hence the nodes in $Residual(N_Y)$ incur zero cost. Thus, when P_r is executed in spill-mode, the budget is sufficiently large to either learn the exact selectivity of Y and X (if the spill-mode execution goes to completion) or to conclude that $q_a \succ q$ (strictly dominates) for some $q \in IC_i$ which implies that $cost(P_{q_a}, q_a) > CC_i$ i.e it lies beyond the contour by PCM. □ Let there be $m = \log_2\left(\frac{C_{max}}{C_{min}}\right)$ number of contours, let P_i be the best 1-plan replacement with sub-optimality $\lambda^{e_1}_i$ for each contour IC_i from $i = 1 \rightarrow m$. Let $\lambda^{e_1} = \max_{i=1\rightarrow m} \lambda^{e_1}_i$.

Lemma 4.2 *The MSO for the 2D scenario when contour plan replacement is done along a single dimension e_1 is $4(1+\lambda^{e_1})$.*

Proof:

The query processing algorithm executes the best 1-plan replacement, P_i, for each contour IC_i, starting from the least cost contour. Each execution of P_i is performed with an increased budget of $CC_i(1+\lambda^{e_1})$. Since each contour now has only 1 plan with fixed inflated budget,

using the PlanBouquet algorithm it is easy to show that the MSO for the 2D scenario is equal to $4(1 + \lambda^{e_1})$.

□

It is important to note that λ^{e_1} - which denotes the worst case sub-optimality incurred for making plan replacements along the dimension e_1 is a function of the dimension e_1 itself.

4.1.1.2 3-D Scenario

In this sub-section we see how the Contour Plan Replacement technique can be extended to the 3-D scenario, consisting of dimensions X, Y and Z. Let us, without loss of generality say that plan replacement along dimension X is to be done. As in the 2D scenario, all the plans on the contour become either Y-spilling or Z-spilling by considering ignoring the epp X. Let the set of plans which were originally X-spilling plans, but now considered as either Y-spilling or Z-spilling, be denoted by P^T.

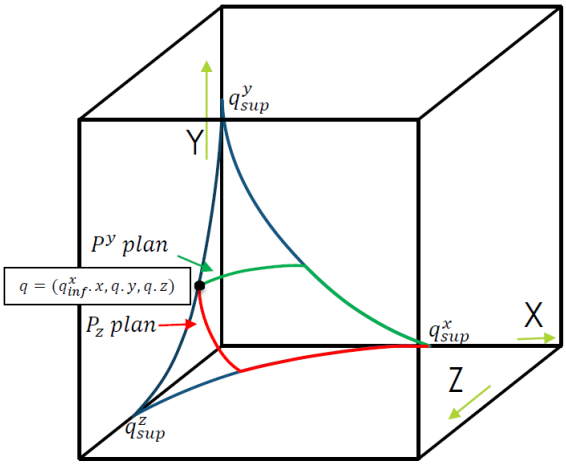


Figure 4.2: Choice of a Plan for a point q

Contour Plan Replacement Strategy : The main idea of the algorithm as stated earlier is to execute two plans (one for each strong dimension) and piggy back the required execution of the weak dimension along with these strong ones. As can be seen in the 2D scenario, the execution of the replaced X dimension is piggy backed with the strong Y dimension. Similarly in the 3D scenario, the X dimension is piggy backed with both the remaining strong dimensions. Now let us now see how to achieve this through contour replacement technique.

Let us first characterize the contours based on the minimum and maximum selectivities of the replaced dimension X, captured by $X = q^x_{inf}.x$ and $X = q^x_{sup}.x$ respectively. There are three possible combinations: a) a 2D contour line on $X = q^x_{inf}.x$ slice and a point on $X = q^x_{sup}.x$ slice; b) a 2D contour line on $X = q^x_{sup}.x$ slice and a point on $X = q^x_{inf}.x$ slice; c) a 2D contour line

on both $X = q^x_{inf}.x$ and $X = q^x_{sup}.x$ slices. The figures in this section correspond to case a) but all the Lemmas and Theorems are generalizable for all the cases mentioned above.

We will first show that how X's execution can be piggy backed with Y. Consider a point q' on the $X = q^x_{inf}.x$ slice, let its coordinates be such that $q' = (q^x_{inf}.x, q'.y, q'.z)$. First we consider the set $S_{q'.y} := \{q | q \in IC_i \text{ and } q.y \leq q'.y\}$ and show that all the (x, y) selectivity combinations such that $y \leq q'.y$ gets covered by a single plan execution. For this single execution, consider the minimal (x, y)-dominating set points of $S_{q'.y}$ denoted by $S^D_{q'.y}$. Formally,

$$S^D_{q'.y} := \forall q \in S_{q'.y}, \exists \hat{q} \in S^D_{q'.y} \text{ such that } (\hat{q}.x, \hat{q}.y) \succeq (q.x, q.y) \tag{4.1}$$

We do a single plan replacement for the set $S^D_{q'.y}$, then by PCM all the (x, y) selectivity combinations of the set $S_{q'.y}$ get covered. Similarly, we can try to piggy back the X's execution with $S_{q'.z}$.

Now let us see how to do two executions for each of the strong dimensions i.e. Y and Z. For every distinct Y and Z values on the contour, we compute the penalty of the sets S^D_y and S^D_z, and assign the minimum of them to each combination (y, z) (shown in Figure 4.2). In essence, every contour point gets a Y-spilling plan or a Z-spilling plan assigned. We then choose the Y-spilling plan and Z-spilling with their respective maximum learning potential as defined in [1]. Let these two maximum respective selectivities be $q^y_{max}.y$ and $q^z_{max}.z$, with the corresponding plans be P^y_{max} ad P^z_{max}. This is shown in Figure 4.3. Then the following lemmas show that the execution of these two max. plans are sufficient to cross the contour.

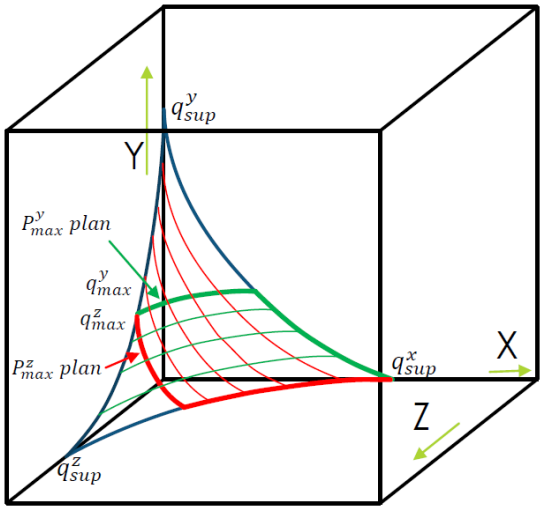


Figure 4.3: Choosing the P^y_{max} and P^z_{max} plans

Lemma 4.3 *Let the following sets be defined as* $S_{q^y_{max}.y} := \{q | q \in IC_i \text{ and } q.y \leq q^y_{max}.y\}$ *and*

$S_{q^z_{max}.z} := \{q | q \in IC_i \text{ and } q.z \leq q^z_{max}.z\}$. *Then every point* $q \in IC_i$ *belongs to either* $S_{q^y_{max}.y}$ *or* $S_{q^z_{max}.z}$.

Proof: Let us prove by contradiction. We know that q^y_{max} is the point on $X = q^x_{inf}.x$ slice which is covered by P^y_{max}, let its coordinates be $q^y_{max} := (q^x_{inf}.x, q^y_{max}.y, z)$. Analogously let $q^z_{max} := (q^x_{inf}.x, y, q^z_{max}.z)$. It is evident that $q^y_{max}.z \leq q^z_{max}.z$, also $q^z_{max}.y \leq q^y_{max}.y$. Hence the point $\bar{q} := (q^x_{inf}.x, q^y_{max}.y, q^z_{max}.z)$ is such that $\bar{q} \succeq q^y_{max}$ and $\bar{q} \succeq q^z_{max}$ which implies that $cost(\bar{q}) \geq CC_i$. Now if there exists a point $\tilde{q}$ such that $\tilde{q} \in IC_i$ but $\tilde{q} \notin S_{q^y_{max}.y}$ and $\tilde{q} \notin S_{q^z_{max}.z}$. This implies that $\tilde{q} \succ \bar{q}$ which further implies $cost(\tilde{q}) > cost(\hat{q}) \geq CC_i$. Hence $\tilde{q} \notin IC_i$. Hence the proof. □

Hence $(S_{q^y_{max}.y} \cup S_{q^z_{max}.z})$ covers all the points of the contour IC_i. This is depicted in Figure 4.4.

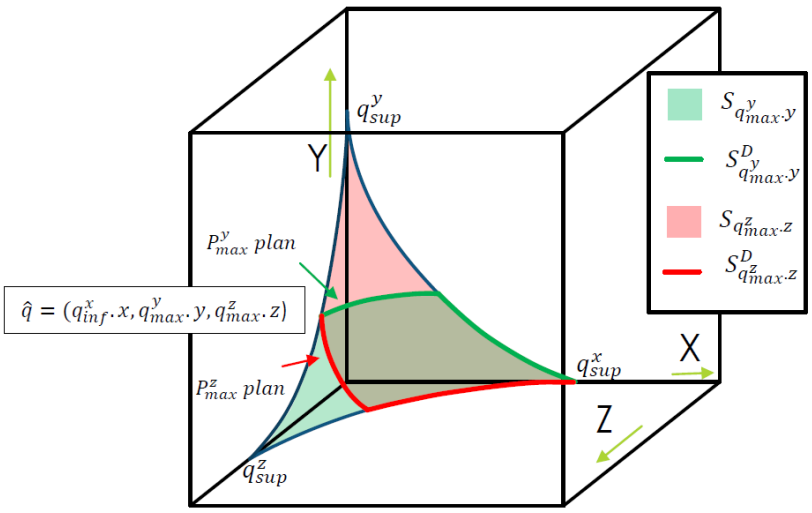


Figure 4.4: Sets $S_{q^y_{max}.y}$, $S_{q^z_{max}.z}$, $S^D_{q^y_{max}.y}$ and $S^D_{q^z_{max}.z}$

Lemma 4.4 *If* P^y_{max} *plan is costed on the set* $S^D_{q^y_{max}.y}$ *with* λ^X_y *as maximum sub-optimality and if* P^y *is executed in spill-mode with budget* $(1+\lambda^X_y)CC_i$ *and does not complete then* $q_a \notin S_{q^y_{max}.y}$.

Proof: Since P^y_{max} plan is costed on the set $S^D_{q^y_{max}.y}$ with λ_y as maximum sub-optimality, and by definition of the set $S^D_{q^y_{max}.y}$, the plan P^y_{max} essentially covers all the combinations of X and Y selectivity pairs for all the points q belonging to contour IC_i such that $q.y \leq q^y_{max}.y$. This is precisely the set $S_{q^y_{max}.y}$. Hence if the point $q_a \in S_{q^y_{max}.y}$ then the spill-mode execution with plan P^y_{max} completes (from Lemma 4.1) and thereby the lemma follows. □

Similarly we can prove the following Lemma,

Lemma 4.5 *If P^z_{max} plan is costed on the set $S^D_{q^z_{max}.z}$ with λ^X_z as maximum sub-optimality and if P^z is executed in spill-mode with budget $(1+\lambda^X_z)CC_i$ and does not complete then $q_a \notin S_{q^z_{max}.z}$.*

Lemma 4.6 *If the spill-mode executions of both the plans, P^y_{max} with budget $(1+\lambda^X_y)CC_i$ and P^z_{max} with budget $(1+\lambda^X_z)CC_i$, do not complete then q_a lies beyond the contour IC_i.*

Proof: From Lemmas 4.4 and 4.5 we can infer that $q_a \notin S_{q^y_{max}.y}$ and $q_a \notin S_{q^z_{max}.z}$. This implies $q_a \notin (S_{q^y_{max}.y} \cup S_{q^z_{max}.z})$ and from Lemma 4.3 we can conclude that q_a lies beyond the contour. □

Consider the situation where q_a is located in the region between IC_k and IC_{k+1}, or is directly on IC_{k+1}. Then, the SpillBound algorithm explores the contours from 1 to k + 1 before discovering q_a. In this process,

Lemma 4.7 *In 3D-scenario the Contour Replacement Strategy ensures that at most two plans are executed from each of the contours $IC_1, ..., IC_{k+1}$, except for one contour in which at most three plans are executed.*

Proof: Let the exact selectivity of one of the epps(Y or Z) be learnt in contour IC_h, where $1 \leq h \leq k+1$. We know that at most two plans are required to be executed in each of the contours $IC_1, \ldots, IC_h$ (from Lemma 4.6). Subsequently, once the selectivity of one of the epps is learnt it boils down to 2-D scenario of Contour Plan Replacement which begins operating from contour IC_h, resulting in three plans being executed in IC_h, and one plan each in contours IC_{h+1} through IC_{k+1}. □

Let $\lambda^X_{IC_i} = \max(\lambda^X_y, \lambda^X_z)$ for each of the contours from $i = 1 \rightarrow m$. Also let $\lambda^X = \max_{\{i=1 \rightarrow m\}} \lambda^X_{IC_i}$.

We now analyze the worst-case cost incurred by SpillBound after the Contour Plan Replacement strategy. For this, we assume that the contour with three plan executions is the costliest contour IC_{k+1}. Since the ratio of costs between two consecutive contours is 2, the total cost incurred by `SpillBound` is bounded as follows: $TotalCost \leq 2 * CC_1(1+\lambda^X) + ... + 2 * CC_k(1+\lambda^X) + 3 * CC_{k+1}(1+\lambda^X)$

$$= (1+\lambda^X)(2 * CC_1 + .. + 2 * 2^{k-1}CC_1 + 3 * 2^k CC_1)$$

$$= (1+\lambda^X)(2 * CC_1(1 + 2 + ...2^k) + 2^k * CC_1)$$

$$= (1+\lambda^X)(2 * CC_1(2^{k+1} - 1) + 2^k * CC_1)$$

$$\leq (1+\lambda^X)(2^{k+2} * CC_1 + 2^k * CC_1)$$

$$= (1 + \lambda^X) * 5 * 2^k * CC_1$$

From the PCM assumption, we know that the cost for an oracle algorithm (that a priori knows the location of q_a) is lower bounded by CC_k. By definition, $CC_k = 2^{k-1} * CC_1$. Hence,

$$MSO \leq \frac{5 * 2^k * CC_1 * (1 + \lambda^X)}{2^{k-1} * CC_1} = 10(1 + \lambda^X)$$

leading to the theorem:

Theorem 4.1 *With the Contour Plan Replacement done along dimension X, the MSO for the 3-D Scenario is* $10(1 + \lambda^X) = MSO_{SB}(2) * (1 + \lambda^X)$.

For the CPR technique to be able to improve on the MSO, we require that $MSO_{SB}(2) * (1 + \lambda^X) < MSO_{SB}(3)$. This is usually the case when X is a weak dimension.

Chapter 5

Results and Observations

In this section, we present the results and observations of the performance, effectiveness and overheads of the techniques discussed in the previous sections. The experimental framework, which is similar to that used in [1], is described first, followed by analysis of the results.

5.1 Database and System Framework

Our experiments are carried out on a representative set of complex OLAP queries, it is comprised of 21 SPJ queries from the TPC-DS benchmark, operating at the base size of 100 GB. The number of relations in these queries range from 4 to 10, and a spectrum of join-graph geometries are modeled, including chain, star, branch, etc.

In order to conduct the assessment on challenging multi-dimensional ESS spaces, we ensure that: (a) All the join predicates and base filter predicates form the set of raw dimensions for the query, (b) The physical schema features indexes on all the attribute columns appearing in the query, maximizing the range of cost values, and hence the number of contours, in the ESS.

The database engine used in our experiments is a modified version of the PostgreSQL 9.4 [7] engine, with the primary additions being:

1. Selectivity Injection, required to generate the ESS,

2. Abstract Plan Costing, used by the optimizer to cost a particular plan at a particular query location,

3. Abstract Plan Execution, required to instruct the execution engine to execute a particular plan and

4. Time-limited execution, required to implement the calibrated sequence of plan executions with associated time budgets.

All the experiments are done on a 16-core HP Z440 workstation with 15 MB L_3 cache, 32 GB RAM and 512 GB SSD.

5.2 Schematic and Projection Removal Results

Using the Schematic and Projection Removal Techniques presented in Sections 3.1 and 3.2 we were able to remove essentially all the base predicates from the 21 TPC-DS queries on which we conducted our experiments. The reduction of the dimensions is depicted in Figure 5.1. The graph also shows an interesting fact, that for all queries the reduced dimensionality of the queries is less than or equal to 5.

It is important to note that the computational effort reduces exponentially with the removal every dimension since compile-time effort is a exponential function in D. Using the Perimeter `MaxInflationFactor` algorithm the effort to identify, calculate the `MIF` and remove these dimensions reduces from $O(r^D) \rightarrow O(2^{(D-k)} * (D-k) * r)$ when k dimensions are removed from the ESS.

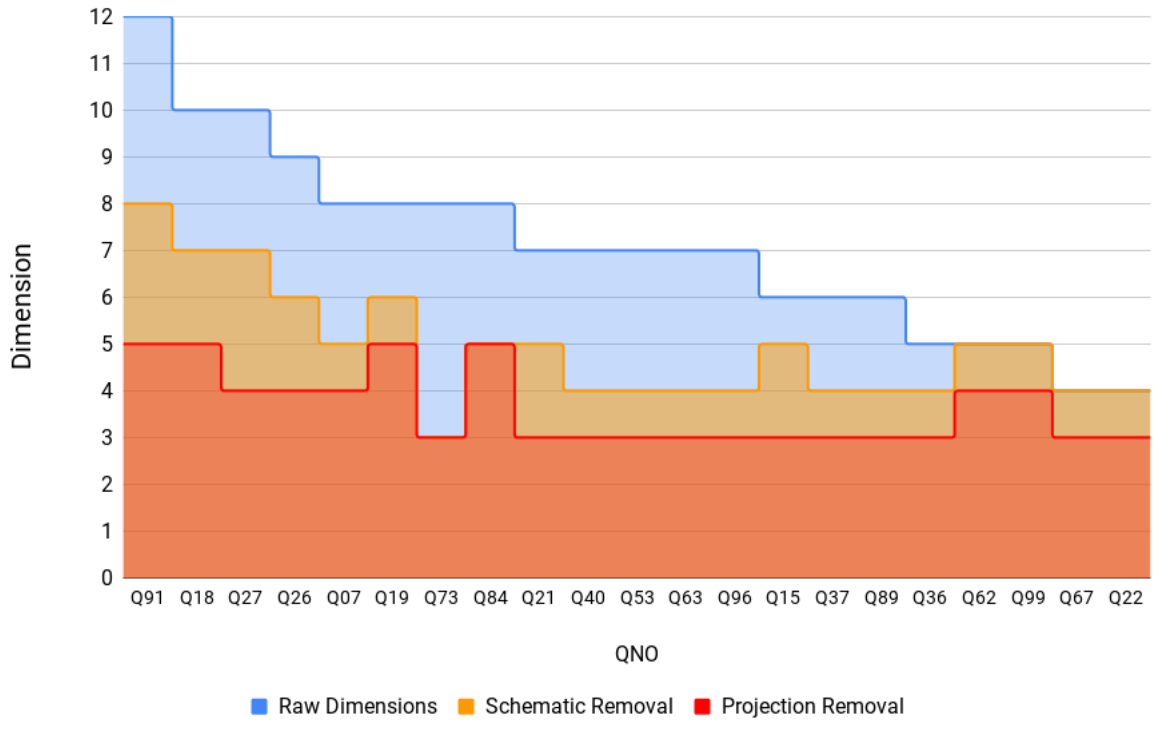
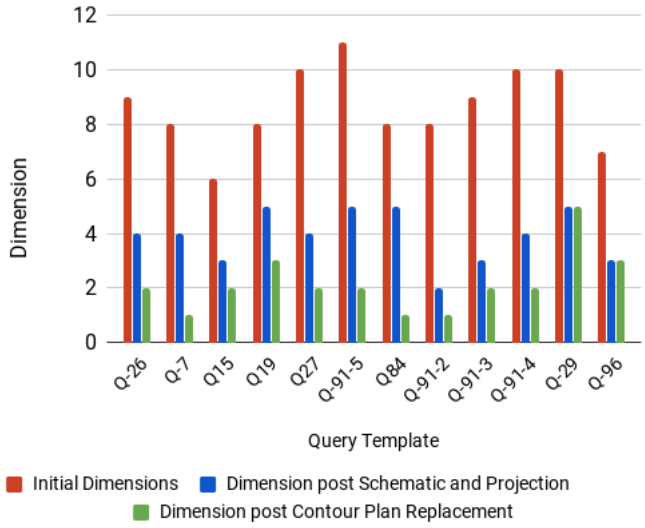
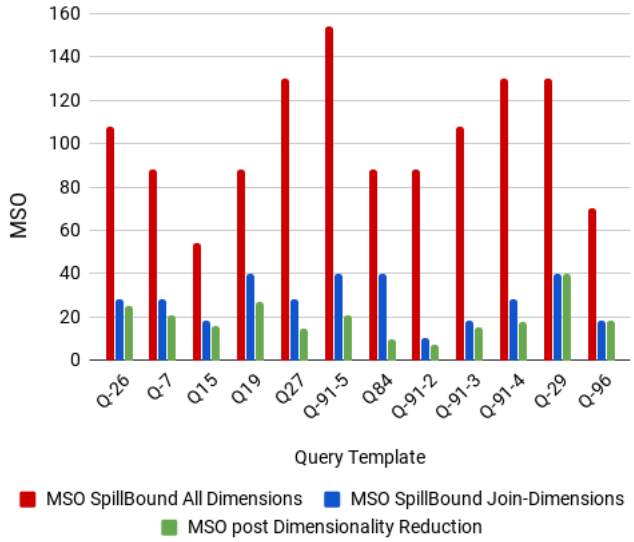


Figure 5.1: Comparison of All Dimensions vs Dimensions post Schematic and Projection Removal

5.3 Contour Plan Replacement Results



(a) Comparison of All Dimensions vs Dimensions post Schematic and Projection Removal vs Dimensions post Contour Plan Replacement



(b) Comparison of MSO SpillBound All Dimensions vs MSO SpillBound Join-Dimensions vs MSO post Dimensionality Reduction

Figure 5.2: Results for Contour Plan Replacement

Figure 5.2a shows the dimensionality reduction in phases for the queries considered in `SpillBound`[1].

The dimensions along which Contour plan replacement is done are virtually removed, this helps in bettering the MSO guarantees only.

For the `SpillBound` queries, we were able to do contour plan replacement along a number of dimensions lowering the MSOs of many queries by making them a function of only the strong dimensions as shown in Figure 5.2b.

Contour Plan Replacement is successful for all the queries except Q-29 and Q-96. This is because the `MaxInflationFactor` for all the dimensions of these queries is very high and hence change in their selectivity causes significant changes in the cost which makes plan replacement with low sub-optimalities difficult along them.

Chapter 6

Conclusions and Future Work

Overall we present a set of techniques for dimensionality reduction of the ESS such that the MSO guarantees are preserved or bettered and the compile-time computational overheads are significantly reduced.

The take-away from this work is that the importance of a dimension is in essence the degree to which it impacts the cost of processing the query, relative to other dimensions, as its selectivity is varied. Our techniques exploit exactly this fact, for instance intuitively the `MaxInflationFactor` for a dimension is low if the PIC exhibits a flat behavior when its selectivity is varied.

If the optimal cost surface does change noticeably when the selectivity of a candidate dimension is varied then removing it using projection may worsen the MSO. But if this change in cost is small relative to other dimensions then it is very likely that the plan structure and join order remains the same (or minor changes) irrespective of the variation in the dimension's selectivity. This is exactly what Contour Plan Replacement leverages when it makes plan replacement along weak dimensions to piggyback the execution of weak dimensions along the strong dimensions. This now makes the MSO of `SpillBound` algorithm a function of only the strong dimensions modulo the penalty incurred for plan replacement.

Our future work would be to make contour plan replacement feasible without the computational effort invested in ESS discovery for weak dimensions.

Bibliography

[1] S.Karthik, J.Haritsa, S.Kenkre and V.Pandit. Platform-independent robust query processing. *In IEEE ICDE Conf.*, 2016. ii, 1, 7, 13, 26, 30, 34, 36

[2] A.Dutt and J.Haritsa. Plan bouquets: Query processing without selectivity estimation. *In ACM SIGMOD Conf.*, 2014. 1

[3] A. Hulgeri and S. Sudarshan. Parametric query optimization for linear and piecewise linear cost functions. *In VLDB*, 2002. 8, 20

[4] D. Harish, P. Darera, and J. Haritsa. Identifying robust plans through plan diagram reduction. *In PVLDB*, 2008. 16

[5] Wang D., Ding C. and Li T. K-Subspace Clustering. *In ECML-PKDD*, 2009. 22

[6] Microsoft SQL Server. https://www.microsoft.com/en-in/sql-server/sql-server-2017 10

[7] PostgreSQL. https://www.postgresql.org/docs/ 10, 34

[8] TPC-H. http://www.tpc.org/tpch/. 1

[9] TPC-DS. http://www.tpc.org/tpcds/. 4