Dimensionality Reduction Techniques for Robust Query Processing

Sanket Purandare Srinivas Karthik Jayant Haritsa

Technical Report
TR-2018-02

Database Systems Lab
Computational and Data Science Dept.
Indian Institute of Science
Bangalore 560012, India

`http://dsl.cds.iisc.ac.in`

Abstract

Selectivity estimation errors have been a long-standing and crippling problem in database query optimization. Over the past few years, a new breed of robust query processing algorithms that completely side-steps this chronic limitation has been developed. Specifically, the brittle estimation procedure is replaced with a *selectivity discovery* process that lends itself to provable worst-case performance guarantees. The new approach is predicated on constructing, at compile-time, a multi-dimensional `PSS` (Predicate Selectivity Space), where each dimension corresponds to a query predicate, and then creating a POSP (Parametric Optimal Set of Plans) overlay on this space. Subsequently, at run-time, a calibrated sequence of time-limited executions from a carefully chosen subset of POSP plans, called the "plan bouquet", takes the query to completion within a bounded response-time, relative to the optimal plan.

Notwithstanding its welcome robustness, the bouquet technique suffers from the "curse of dimensionality" on two fronts – firstly, the overheads of constructing the POSP overlay are *exponential* in the `PSS` dimensionality, and secondly, the performance guarantees are *quadratic* in the `PSS` dimensionality. Since contemporary OLAP queries often have a high ab initio `PSS` dimensionality, the practicality of the bouquet technique is immediately called into question. We tackle this issue here and present a principled and efficient three-stage pipeline, called `DimRed`, that incorporates *schematic*, *geometric* and *piggybacking* strategies, to achieve sizeable reductions in `PSS` dimensionality.

`DimRed` has been empirically evaluated on a large suite of representative queries sourced from the TPC-DS and JOB benchmarks. Our results indicate that the effective dimensionality can be consistently brought down to *five* or less. The quantitative benefits of this reduction are orders of magnitude of reduction in the compile-time overheads, and an average improvement of around two-thirds in the performance guarantees. Therefore, in an overall sense, `DimRed` offers a substantive step forward in making robust query processing feasible on current environments.

1 Introduction

The traditional approaches for optimizing declarative OLAP queries (e.g. [21, 7]) are contingent on estimating a host of *predicate selectivities* in order to find the optimal execution plan. For instance, even the toy TPC-H query shown in Figure 1, which lists the details of cheap parts, requires *three* selectivities to be estimated (one filter predicate and two join predicates).

```
select * from lineitem, orders, part where p_partkey = l_partkey
    and o_orderkey = l_orderkey and p_retailprice < 1000
```

Figure 1: Example TPC-H Query

Unfortunately, in practice, these estimates are often significantly in error with respect to the values actually encountered during query execution, leading to poor plan choices and grossly inflated response times [18]. The severity of this problem in industrial settings has also been highlighted in a series of Dagstuhl Seminars on Robust Query Processing during the last decade [1, 2, 3], and in the recent database literature (e.g. [10]).

Over the past few years, a new breed of robust query processing algorithms that completely side-steps the above chronic limitation has been developed. Specifically, the brittle estimation

procedure is replaced with a *selectivity discovery* process that lends itself to provable worst-case performance guarantees. The initial concepts for this approach were presented in the `PlanBouquet` algorithm [5, 6], and subsequently refined in the `SpillBound` algorithm [12, 14].

1.1 Selectivity Discovery with `SpillBound`

The `SpillBound` (`SB`) algorithm initially constructs, at query compile-time, a D-dimensional "Predicate Selectivity Space" (`PSS`), where each dimension corresponds to a query predicate whose selectivity needs to be discovered. An example 3D selectivity space, corresponding to the three-predicate query in Figure 1, is shown in Figure 2, with each dimension ranging over $[0, 1]$.

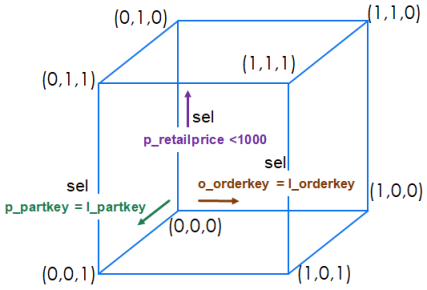


Figure 2: Example 3D `PSS`

The next step is to identify, through repeated invocations of the query optimizer, the "parametric optimal set of plans" (POSP) that cover the entire selectivity space contained in the `PSS`. This overlay exercise is carried out at a discretized resolution r along each dimension of the `PSS`, incurring a total of $\Theta(r^D)$ calls to the query optimizer.[1]

At run-time, starting from the origin of the `PSS` and moving outwards in the space, a carefully chosen subset of POSP plans, called the "plan bouquet", is sequentially executed, with each execution assigned a time limit equal to the plan's optimizer-assigned cost. The choice of plans is such that each execution focuses on incrementally learning the selectivity of a specific predicate. This sequence of plan executions ends when all the selectivities in the `PSS` have been fully discovered. Armed with this complete knowledge, the genuine optimal plan is now correctly identified via the query optimizer, and used to finally execute the query to completion.

A perhaps surprising outcome of `SpillBound`'s plan-switching strategy is that the additional execution costs incurred during the progressive discovery process can be *bounded* relative to the optimal, *irrespective of the query location in the space*. Specifically, let us use Maximum Sub-Optimality (**MSO**), as defined in [6], to capture the *worst-case* sub-optimality ratio of a query processing algorithm over the entire D-dimensional `PSS`, relative to an oracle that magically knows the correct query location and therefore directly uses the ideal plan. Then, the MSO of `SpillBound` has the following upper bound:

$$MSO_{SB}(D) \leq D^2 + 3D \tag{1}$$

This bound represents the first-such guarantee in the literature. Moreover, its numerical values are very low in comparison to the *orders of magnitude* slowdowns reported for contemporary query optimizers [18, 17, 19].

[1] Strictly speaking, `SpillBound` does not require characterization of the entire `PSS`, but the exponential dependency on D remains even in the optimized versions [6].

Notwithstanding the unique and welcome benefits of SpillBound with regard to robust query processing, it suffers from the *"curse of dimensionality"* on two important fronts – firstly, the overheads of constructing the POSP overlay are *exponential* in the PSS dimensionality, and secondly, the MSO guarantees are *quadratic* in this dimensionality. Since contemporary OLAP queries often have a high ab initio PSS dimensionality, a legitimate question that arises is whether the bouquet technique can be made practical for current database environments. As a case in point, consider the SPJ version of TPC-DS Query 27 shown in Figure 3, whose raw dimensionality is **9** (comprised of 4 join predicates, 4 equality filter predicates, and 1 set membership predicate). Constructing its PSS at even a modest resolution of $r = 20$ (corresponding to 5% increments in the selectivity space) would require making about 0.5 *trillion* calls to the query optimizer, and the MSO would exceed *100*.

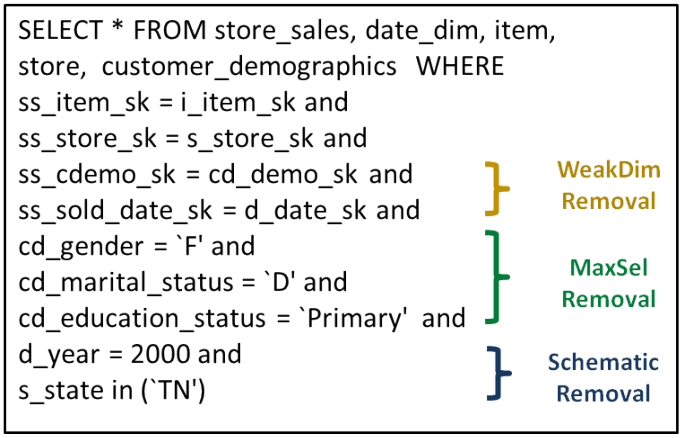


Figure 3: TPC-DS Query 27 (SPJ version)

In the research heretofore on plan bouquets, the PSS dimensionality issue was handled by manually identifying and eliminating dimensions that were either accurately estimated by the optimizer, or whose errors did not materially impact the overall plan performance – the resulting reduced space was termed as the Error Selectivity Space (ESS). This inspection-based approach is not a scalable solution, and moreover, may have missed opportunities for dimension removal due to its ad-hoc nature. We have therefore investigated the development of automated techniques for converting high-dimensional PSS into equivalent low-dimensional ESS, and report on the outcomes here.

1.2 Problem Definition

Given the above framework, a mandatory criterion for our dimensionality reduction algorithm is that the MSO of the resultant ESS should be no worse than that of the initial PSS – that is, the reduction should be *"MSO-safe"*. Within this constraint, there are two ways in which the optimization problem can be framed – we can choose to either minimize the compilation overheads, or to minimize the MSO, leading to the following problem definitions:

Overheads Metric: Develop an MSO-safe time-efficient ESS construction algorithm that, given a query Q with its PSS, removes the maximum number of PSS dimensions.

MSO Metric: Develop an MSO-safe time-efficient ESS construction algorithm that, given a query Q with its PSS, removes a set of PSS dimensions such that the resulting MSO is minimized.

1.3 The DimRed Procedure

In this paper, we tackle the above-mentioned PSS dimensionality problem. Specifically, we present a principled and efficient process, called DimRed, that incorporates a pipeline of reduction strategies whose collective benefits ultimately result in ESS dimensionalities that can be efficiently handled by modern computing environments.

The DimRed procedure is composed of three components: SchematicRemoval, MaxSelRemoval, and WeakDimRemoval that are applied in sequence in the processing chain from the user query submission to its execution with SpillBound, as shown in Figure 4. Here, SchematicRemoval and MaxSelRemoval reduce the overheads through *explicit* removal of PSS dimensions, whereas WeakDimRemoval improves the MSO through *implicit* removal of dimensions. To illustrate DimRed's operation, we use TPC-DS Query 27 (Figure 3) as the running example.

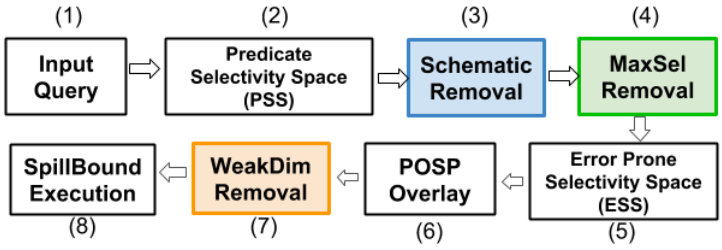


Figure 4: DimRed Pipeline

In the first component, SchematicRemoval, a dimension d is removed whenever we expect that the selectivity estimates made by the optimizer, using either the metadata statistics or the physical schema, will be highly accurate. For instance, database engines typically maintain exact frequency counts for columns with only a limited number of distinct values in their domains – therefore, the dimensions corresponding to the d_year and s_state columns in Q27 can be safely removed from the PSS.

The second component, MaxSelRemoval, takes a cost-based approach to identify "don't-care" dimensions where the actual selectivity value does not play a perceptible role on overall performance. Specifically, given a candidate dimension d, it conservatively assumes that dimension d's selectivity is the *maximum* possible (typically, 1). Due to this movement to the ceiling value, there are countervailing effects on the MSO guarantee – on one hand, the value of D is decremented by one in Equation 1, but on the other, an *inflation factor* α_d is suffered due to SpillBound's choice of bouquet plan executions being now dictated by the maximum selectivity, rather than the actual value. The good news is that α_d can be bounded and efficiently computed, as explained in Section 4. Therefore, we can easily determine whether the benefits outweigh the losses, and accordingly decide whether or not to remove a dimension. For instance, in Figure 3, the three filter predicates on cd_gender, cd_marital_status and cd_education_status can be removed since their inflation factors are small, collectively amounting to just 1.34.

After the explicit dimension removal by the SchematicRemoval and MaxSelRemoval components, the POSP overlay of the ESS is computed. Subsequently, rather than separately discovering the selectivity of each of the remaining dimensions, we attempt, using the WeakDimRemoval component, to *piggyback* the selectivity discovery of relatively "weak" dimensions on their "strong" siblings – here, the strength of a dimension is characterized by its α_d value, as computed previously by the MaxSelRemoval module. That is, a dimension d with low α_d is discovered concurrently with a high inflation counterpart – again, there are countervailing factors since the concurrent discovery effectively reduces the dimensionality by one, but incurs a second inflation

factor β_d due to the increased budgetary effort incurred in this process. However, the good news again is that β_d can be bounded and efficiently computed if the ESS is available, as explained in Section 5, and it can be easily determined whether the benefits outweigh the losses. For instance, in Figure 3, the last two join predicates are implicitly removed through this process, since their execution is piggybacked on the first two join predicates.

1.4 Performance Results

The summary theoretical characterization of the DimRed procedure, with regard to dimensionality, computational overheads and MSO guarantees, is captured in Table 1. In this table, k_s and k_m denotes the number of dimensions that are explicitly removed thanks to the SchematicRemoval and MaxSelRemoval components, respectively, while k_w denotes the number of implicitly removed dimensions from the WeakDimRemoval component. α_M captures the collective MSO inflation factor arising from the removal of the k_m don't-care dimensions, while β_W indicates the net inflation factor arising out of the k_w piggybacked discoveries. The last column compares the cumulative overheads incurred after applying the DimRed pipeline relative to what would have been incurred on the native PSS.

Table 1: Summary Performance Characterization

	Dimensionality	**Maximum Suboptimality**	**Overheads (Opt Calls)**
PSS	D	$MSO_{\text{SB}}(D)$	r^D
Schematic Removal	$D - k_s$	$MSO_{\text{SB}}(D - k_s)$	$r^{D-k_s-k_m} + \theta(2^{D-1} * D * r)$
MaxSel Removal	$(D - k_s - k_m)$	$\alpha_M * MSO_{\text{SB}}(D - k_s - k_m)$	
WeakDim Removal	$(D - k_s - k_m - k_w)$	$\alpha_M * \beta_W * MSO_{\text{SB}}(D - k_s - k_m - k_w)$	

Given this characterization, we need to assess whether the values of k_s, k_m and k_p are substantial enough in practice to result in a low-dimensional ESS, and we have therefore conducted a detailed empirical evaluation of the DimRed procedure. Specifically, we have evaluated its behavior on a representative suite of 50-plus queries, sourced from the popular TPC-DS and JOB benchmarks. Our results indicate that DimRed is consistently able to bring down the PSS dimensionality of the workload queries, some of which are as high as 12, to 5 or less. A sample outcome for Query 27 is shown in Table 2, and we see here that the original dimensionality of 9 is brought down to as low as 1 when optimizing for overheads, and as low as 2 when optimizing for MSO. Further, the preprocessing time taken before query execution can actually begin is now down to seconds from days. Finally, the resulting MSOs are not only safe, but significantly better than those on the original system – for instance, optimizing for MSO produces a huge improvement from 108 to less than 20.

Organization.

The remainder of this paper is organized as follows: The problem framework and notations are formalized in Section 2. The three components of DimRed, namely, SchematicRemoval, MaxSelRemoval and WeakDimRemoval, are presented and theoretically analyzed in Sections 3, 4 and 5, respectively. The experimental framework and performance results are highlighted in Section 6. Finally, our conclusions are summarized in Section 7.

Table 2: Results for TPC-DS Q27

	Overheads Metric			MSO Metric		
	Retained Dimensions.	**MSO**	**Overheads (Opt calls)**	**Retained Dimensions.**	**MSO**	**Overheads (Opt calls)**
PSS	9	108	0.5 trillion	9	108	0.5 trillion
Schematic Removal	7	70	40700 (9.2 secs)	7	70	202300 (45 secs)
MaxSel Removal	1	70		4	37.5	
WeakDim Removal	-	-		2	20	

2 Problem Framework

We begin by reviewing the key concepts and assumptions underlying our approach to robust query processing [6, 12].

2.1 Selectivity Spaces

Given an SQL query, any predicate for which the optimizer invokes the selectivity estimation module is referred to as a *selectivity predicate*, or sp. For a query with D sps, the set of all sps is denoted by SP $= \{s_1, \ldots, s_D\}$, where s_j denotes the j^{th} sp. The selectivities of the D sps are mapped to a D-dimensional space, with the selectivity of s_j corresponding to the j^{th} dimension. Since the selectivity of each predicate ranges over $[0, 1]$, a D-dimensional hypercube $[0, 1]^D$ results, henceforth referred to as the *predicate selectivity space* (PSS). Note that each location $q \in [0, 1]^D$ in the PSS represents a specific query instance where the sps happen to have the selectivities corresponding to q. Accordingly, the selectivity value of q on the j^{th} dimension is denoted by $q.j$.

For tractability, the PSS is discretized at a fine-grained resolution r in each dimension. We refer to the location corresponding to the minimum selectivity in each dimension as the *origin* of the PSS, and the location at which the selectivity value in each dimension is maximum as the *terminus*.

As shown in this paper, some selectivity dimensions may not be error-prone, and are therefore liable to be removed by SchematicRemoval. Further, some other dimensions may be error-prone, but their errors do not materially impact the overall processing cost, and can therefore be removed by MaxSelRemoval. The dimensions that are retained after these pruning steps are called as *impactful error-prone predicates*, or epp, and they collectively form the Error Selectivity Space (ESS).

2.2 POSP Plans

The optimal plan for a generic selectivity location $q \in$ PSS is denoted by P_q, and the set of such optimal plans over the complete PSS constitutes the *Parametric Optimal Set of Plans* (POSP) [11].[2] We denote the cost of executing any plan P at a selectivity location $q \in$ PSS by $Cost(P, q)$. Thus, $Cost(P_q, q)$ represents the optimal execution cost for the selectivity instance located at q . For ease of presentation, we will hereafter use *cost of a location* to refer to the cost of the optimal plan at that location, and denote it by $\textsc{Cost}(q)$. Finally, we assume that the query optimizer can

[2]Letter subscripts for plans denote locations, whereas numeric subscripts denote identifiers.

identify the optimal query execution plan if the selectivities of all the sps are correctly known.[3]

2.3 Maximum Sub-Optimality (MSO)

We now move on to describing the MSO performance metric used in [12] to quantify the robustness of query processing. For this purpose, let q_a denote the PSS location corresponding to the actual selectivities of the user query epps – note that this location is unknown at compile-time, and needs to be explicitly discovered. SpillBound carries out a sequence of budgeted plan executions in order to discover the location of q_a. We denote this sequence by $\texttt{Seq}_{q_a}$, with each element t_i in the sequence being a pair, (P_i, ω_i) indicating that plan P_i is executed with a maximum time budget of ω_i.

The sub-optimality of this plan sequence is defined relative to an oracle that magically knows the correct query location apriori and therefore directly uses the ideal plan P_{q_a}. That is,

$$SubOpt(\texttt{Seq}_{q_a}) = \frac{\sum_{t_i \in \texttt{Seq}_{q_a}} \omega_i}{\textsc{Cost}(q_a)}$$

from which we derive

$$MSO = \max_{q_a \in \texttt{PSS}} SubOpt(\texttt{Seq}_{q_a})$$

In essence, MSO represents the *worst-case* suboptimality that can occur with regard to plan performance over the entire PSS space.

2.4 Assumptions

The notion of a location q_1 *spatially dominating* a location q_2 in the ESS plays a central role in our robust query processing framework. Formally, given two distinct locations $q_1, q_2 \in \texttt{PSS}$, q_1 spatially dominates q_2, denoted by $q_1 \succ q_2$, if $q_1.j \geq q_2.j$ for all $j \in \{1, \ldots, D\}$. Given spatial domination, an essential assumption that allows SpillBound to systematically explore the PSS is that the cost functions of the plans appearing in the PSS all obey *Plan Cost Monotonicity* (PCM). This constraint on plan cost function (PCF) behavior may be stated as follows: For any pair of distinct locations $q_b, q_c \in \texttt{PSS}$, and for any plan P,

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

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. In a nutshell, *spatial domination implies cost domination.*

2.5 Notations

For easy reference, the notations used in the remainder of the paper are summarized in Table 3.

[3]For example, through the classical Dynamic Programming-based search of the plan enumeration space [21].

Table 3: NOTATIONS

Notation	Meaning
PSS	Predicate Selectivity Space
D	Dimensionality of the PSS
$s_1, ..., s_D$	Selectivity predicates in the query
ESS	Error Selectivity Space
epp (EPP)	Error-prone predicates
$q \in [0,1]^D$	A location in the selectivity space
$q.j$	Selectivity of q in the j^{th} dimension
P_q	Optimal Plan at q
q_a	Actual selectivity of query
$Cost(P,q)$	Cost of plan P at location q
$\text{COST}(q)$	Cost of the optimal plan at location q
α_d	Max cost inflation of MaxSelRemoval for dimension d
β_d	Max cost inflation of WeakDimRemoval for dimension d
α_M	Cumulative cost inflation of MaxSelRemoval
β_W	Cumulative cost inflation of WeakDimRemoval

3 Schematic Removal of Dimensions

We now present an approach for schematic removal of dimensions from the PSS. This component is based on the observation that using standard meta-data structures such as histograms, and physical schema structures such as indexes, it is feasible to establish the selectivities of some of the query predicates with complete or very high accuracy. Further, even if an almost-precise value cannot be established, the metadata could serve to provide tighter lower and upper bounds for selectivities as compared to the default $(0, 1]$ range – these bounds can be leveraged by the subsequent MaxSelRemoval stage of the DimRed pipeline. We hasten to add that while what we describe here is largely textbook material, we include it for completeness and because of its significant reduction impact on typical OLAP queries, as highlighted in our experimental results of Section 6.

For starters, consider the base case of a filter predicate on an ordered domain, a very common occurrence in OLAP queries, whose selectivity analysis can be carried out as follows for equality and range comparisons, respectively.

Equality Predicates: Database engines typically store the *exact* frequency counts for the most commonly occurring values in a column. Therefore, if the equality predicate is on a value in this set, the selectivity estimate can be made accurately. On the other hand, values outside of this set will be associated with some bucket of the column's histogram. Therefore, the selectivity range can be directly bounded within $[0, BucketFrequency]$.

An alternate approach to selectivity estimation is to use, if available, an index on the queried column. This is *guaranteed* to provide accurate estimates, albeit at higher computational cost arising out of index traversal. However, since the typical running times for OLAP queries are in several minutes, investing a few seconds on such accesses appears to be an acceptable tradeoff, especially given that choosing wrong plans due to incorrect estimates could result in arbitrary blowups of the response time.

Range Predicates: In this case, histograms can be easily leveraged to obtain tighter bounds,

especially with equi-depth histogram implementations. Specifically, the lower bound for the selectivity range is the summation of the frequencies of the buckets that entirely fall within the given range, and the upper bound is the summation of the frequencies of the buckets that partially or completely overlap with the given range.

Similar to equality predicates, if an index is available on the predicate column, then accurate estimates are guaranteed while incurring the index traversal costs.

String predicates: The traditional meta-data structures for string queries often do not yield satisfactory accuracy for selectivity estimation, and typically tend to under-estimate the expected cardinalities. Moreover, they are not particularly useful for obtaining tight lower and upper bounds. Hence, we leverage the strategy described in [15], where summary structures based on *q-grams* are proposed for storing string-related metadata. But for combining the selectivities of the individual sub-string predicates and obtaining deterministic tight bounds, we make use of the *Short Identifying Substring Hypothesis* stated and applied in [4, 16]. We also hasten to add that even though these aforementioned strategies provide point estimates for the selectivities, we just adopt their mechanisms for providing bounds on the selectivity range.

The above discussion was for individual predicates. However, in general, there may be multiple filter predicates on a base relation. In such cases, we first compute the ranges or values for each individual predicate (in the manner discussed above), and then use these individual bounds to compute bounds on the relational selectivity as a whole. For instance, when there are conjunctive predicates on a relation, the upper bound on the relational selectivity is simply the upper bound of the least selective predicate. Analogously, for disjunctive predicates, the lower bound is simply the maximum lower bound among the individual predicates. After determining the bounds of the relational selectivity, we vary the individual predicates between $[0, 1]$, discarding any selectivity combinations that violate the relational selectivity bounds.

4 MaxSel Removal of Dimensions

After the schematic removal of dimensions is completed, we know that the optimizer may not be able to accurately estimate the selectivities of the remaining dimensions. However, it may still be feasible to consider some of these selectivities as *"don't-cares"* (i.e. $*$ in regex notation), which is equivalent to removing the dimensions, while provably continuing to maintain overall MSO-safety. The systematic identification and removal of such don't-care dimensions is carried out by the `MaxSelRemoval` module – it does so by the simple expedient of assigning the maximum selectivity (typically, 1) to the candidate dimensions. This maximal assignment *guarantees*, courtesy the PCM assumption (Section 2.4), that the time budgets subsequently allocated by `SpillBound` to the bouquet plans in the reduced space are sufficient to cover *all* selectivity values for these dimensions. What is left then is to check whether these deliberately bloated budgets could result in a violation of MSO-safety – if not, the dimensions can be removed.

We describe the operation of the `MaxSelRemoval` module in the remainder of this section. For ease of understanding, we first consider the baseline case of a 2D `PSS`, and then extend the design to higher dimensions. This is followed by an analysis of the module's algorithmic efficiency. Further, for ease of notations we assume D dimensions are retained post the `SchematicRemoval` phase.

4.1 Baseline Case: 2D Selectivity Space

Consider a 2D PSS with dimensions X and Y, as shown in Figure 5, and let the actual (albeit unknown) selectivity of the query in this space be an arbitrary location $q_a(x, y)$. Now assume that we wish to establish whether the X dimension can be dropped in an MSO-safe manner. We begin by projecting q_a to the extreme X-boundaries of the PSS, that is, to $X = 0$ and $X = 1$, resulting in $q_a^{min} = (0, q_a.y)$ and $q_a^{max} = (1, q_a.y)$, respectively, as shown in Figure 5.

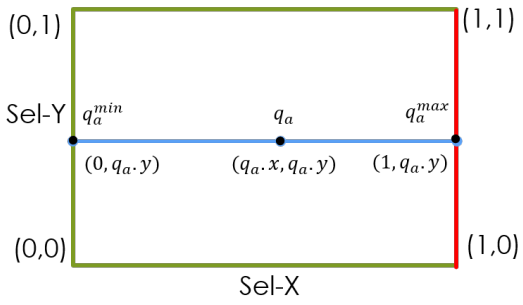


Figure 5: Example 2D PSS

Next, we compute a bound on the sub-optimality of using SpillBound with bloated time budgets based on q_a^{max}.

Lemma 4.1 *The sub-optimality for q_a is at most $4 * \frac{\text{Cost}(q_a^{max})}{\text{Cost}(q_a^{min})}$ after removing dimension X from the 2D PSS.*

Proof Let the total execution cost incurred by the SpillBound algorithm be denoted by $Cost_{SB}(q_a^{max})$. Then, its sub-optimality is given by

$$SO_{SB} = \frac{Cost_{SB}(q_a^{max})}{\text{Cost}(q_a)}$$
$$= \frac{Cost_{SB}(q_a^{max})}{\text{Cost}(q_a^{max})} * \frac{\text{Cost}(q_a^{max})}{\text{Cost}(q_a)}$$

From Equation 1, we know that the MSO of SB for a single dimension is 4, and therefore $\frac{Cost_{SB}(q_a^{max})}{\text{Cost}(q_a^{max})}$ is also upper-bounded by 4. Hence,

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

Now by the PCM assumption, we know that $\text{Cost}(q_a) \geq \text{Cost}(q_a^{min})$. Therefore, we have

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

The ratio $\frac{\text{Cost}(q_a^{max})}{\text{Cost}(q_a^{min})}$ captures the *inflation* in sub-optimality for q_a. But note that q_a can be located anywhere in the 2D space, and we therefore need to find the maximum inflation over *all*

possible values of y for q_a, and this is denoted by α_X. Formally, the α_X for removing dimension X is defined as:

$$\alpha_X := \max_{q_a.y \in [0,1]} \frac{\text{COST}(q_a{}^{max})}{\text{COST}(q_a{}^{min})}$$

This leads us to the generalization:

Corollary 4.2 *After removing a single dimension d from a 2D* PSS*, the MSO increases to at most $4 * \alpha_d$.*

Now all that remains is to check whether MSO-safety is retained in spite of the above inflation – specifically, whether

$$4 * \alpha_d \leq 10$$

If the check is true, dimension d can be safely removed from the PSS after assigning it the maximum selectivity. (The value 10 comes from the MSO bound for 2D in Equation 1.)

Testing Overheads

We now turn our attention to the computational effort incurred in the dimension removal testing procedure. Note that calculating α_d only requires knowledge of the boundaries $Sel(d) = 0$ and $Sel(d) = 1$, and therefore only $4r$ optimization calls are required in total for testing both dimensions in a 2D space. This is in sharp contrast to the r^2 calls that would have been required for a POSP overlay of the complete PSS.

4.2 Extension to Higher Dimensions

We now move on to calculating the maximum inflation in sub-optimality for the generic case of removing k dimensions, $s_1, \ldots, s_k$ from a D-dimensional PSS, while maintaining MSO-safety. For this, the key idea, analogous to the 2D case, is given a q_a in the PSS, find the cost ratios between when the selectivities of the $s_1, \ldots, s_k$ dimensions of q_a are all set to 1, and when they are all set to 0. Since q_a can be located anywhere in the PSS, these cost ratios have to be computed for *all* possible selectivity combinations of the retained dimensions. Finally, the maximum of these values gives us $\alpha_{s_1,\ldots,s_k}$. That is,

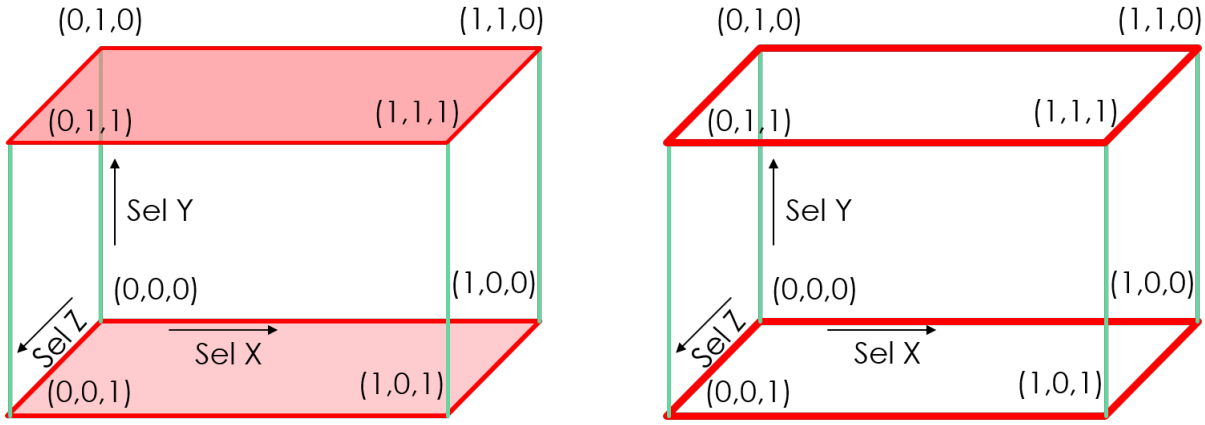
$$\alpha_{s_1,\ldots,s_k} = \max_{\forall (q.j_{k+1},\ldots,q.j_D) \in [0,1]^{D-k}} \frac{Cost(q^{max})}{Cost(q^{min})} \tag{2}$$

with q^{max} and q^{min} corresponding to the locations with $q.i = 1$ and $q.i = 0$, respectively, $\forall i \in \{1, .., k\}$.

To illustrate the above, consider the example 3D PSS with dimensions X, Y and Z shown in Figure 6a. Here, if we wish to consider dimension Y for removal, we need to first compute the sub-optimality inflations across all matching pairs of points in the (red-colored) 2D surfaces corresponding to $Y = 0$ and $Y = 1$. Then, α_Y is given by the maximum of these inflation values.

Finally, to determine whether the removal of the $s_1, \ldots, s_k$ dimensions is MSO-safe, the check is simply the following:

$$\alpha_{s_1,\ldots,s_k} * MSO_{\text{SB}}(D-k) < MSO_{\text{SB}}(D) \tag{3}$$



(a) α_Y from Surface (b) α_Y from Perimeter

Figure 6: 3D PSS - Calculation of α_Y

Testing Overheads:

The computational efforts incurred, in terms of optimizer calls, for calculating $\alpha_{s_1,\ldots,s_k}$ is $\theta(2^D * r^{D-1})$. This is because the suboptimality inflations need to be calculated for *every* selectivity combination of the retained dimensions, making the testing overheads to grow exponentially with the PSS dimensionality. It may therefore appear, at first glance, that we have merely shifted the computational overheads from the exhaustive POSP overlay to the modules of the DimRed pipeline. However, as is shown next, it is feasible, with mild assumptions, to design an efficient mechanism to compute α_M.

4.3 Efficient Computation of MaxSelRemoval

To provide efficiency in the MaxSelRemoval algorithm, we bring in two techniques, *Greedy Removal* and *Perimeter Inflation*. With Greedy Removal, we do not exhaustively consider all possible groups of dimensions for removal. Instead, we first compute for each dimension d, its individual α_d assuming that just d is removed from the PSS and all other dimensions are retained. Based on these inflation values, a sequence of dimensions is created in increasing α_d order. Then, we iteratively consider *prefixes* of increasing length from this sequence with the stopping criterion based on the objective – overheads minimization or MSO minimization.

As an example, we show in Figure 7 the resultant MSOs for the greedy removal of dimensions for TPC-DS Q7, which has an 8-dimensional PSS. As can be seen in the figure, the MSO initially declines from its starting value of 88 as we keep removing the low α_d dimensions. However, after a certain point, it begins to rise again. When overheads minimization is the objective, the algorithm will remove the dimension group {*d4, d5, d6, d7, d0, d1, d2*}, and then stop due to violation of MSO-safety. By this time, the ESS dimensionality is reduced to just 1, and the MSO guarantee is 70. On the other hand, when MSO minimization is the objective, the minimum MSO of 33 is obtained by removing dimensions {*d4, d5, d6, d7*} with a combined inflation factor of just **1.17**!

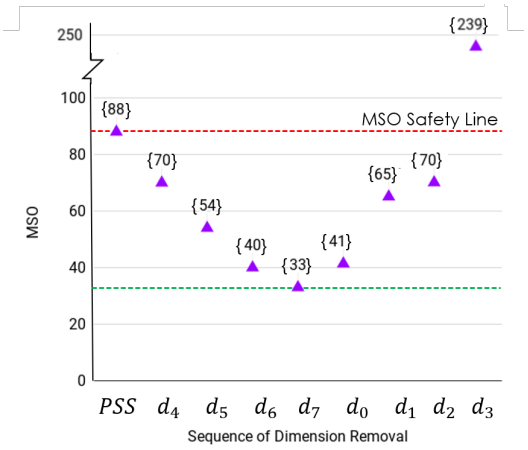


Figure 7: MSO Profile for Greedy Dimension Removal

Although the greedy removal strategy does significantly lower the overheads, it still requires $\mathcal{O}(D * r^{D-1})$ optimizer calls since the α_d has to be calculated for every dimension d. This is where we bring in our second strategy of *Perimeter Inflation*. Specifically, we assume that the α_d (for every dimension d) is always located on the *perimeter* of the PSS and not in the interior – if this is true, then the number of optimizer calls required to calculate for all α_ds is reduced to $\theta(2^{D-1} * D * r)$, which is *linear* in the resolution. We have empirically verified, as highlighted in Section 6, that this assumption is is generally valid. Moreover, in the remainder of this section, we formally prove that, under some mild assumptions, the perimeter location of the α_d is only to be expected.

4.4 Proof of Perimeter Inflation

For ease of exposition, we first analyze a 3D PSS, and later generalize the proof to higher dimensions. (The proof technique used here is similar in spirit to that used in [9] for ensuring bounded suboptimalities of plan replacements in a selectivity space.)

Consider the 3D PSS shown in Figure 6b, with dimensions X, Y and Z, and dimension Y being the candidate for removal. Our objective is to show that optimization calls are required only along the red perimeter lines in the figure, and not along the surface walls (unlike Figure 6a). To start with, we introduce an inflation function, f, that captures the sub-optimality inflation as a function of the selectivity combinations of the retained dimensions X and Z, along the extreme values of the removed dimension Y. Formally, the inflation function, $f(x, z)$, is defined as follows:

$$f(x, z) = \frac{\text{COST}(q.x, q.y = 1, q.z)}{\text{COST}(q.x, q.y = 0, q.z)}$$

Behavior of function f

To analyze f's behavior, we leverage the notion of optimal cost surface (OCS), which captures the cost of the optimal plan at every location in the PSS. For now, assume that the OCS exhibits *axis-parallel piecewise linearity* (APL) in X, Y and Z. That is, the OCS is of the form:

$$OCS(x, y, z) = u_1x + u_2y + u_3z + u_4xy + u_5yz + u_6xz + u_7xyz + u_8 \quad (4)$$

where the u_i are arbitrary scalar coefficients.

When dimension $Y = 1$, the projected OCS with this y value is represented as

$$OCS|_{y=1} = a_1x + a_2z + a_3xz + a_4, \tag{5}$$

where the a_i are the new scalar coefficients. Analogously, when $Y = 0$, the projected OCS becomes

$$OCS|_{y=0} = b_1x + b_2z + b_3xz + b_4 \tag{6}$$

Thus f can now be rewritten as a point-wise division of a pair of 2D APL functions:

$$f(x, z) = \frac{OCS|_{y=1}}{OCS|_{y=0}} = \frac{a_1x + a_2z + a_3xz + a_4}{b_1x + b_2z + b_3xz + b_4} \tag{7}$$

Now consider the function $f_z(x)$, which keeps dimension Z constant at some value and varies only along dimension X. When $Z = z_0$, the various possible behaviors of $f_{z_o}(x)$ are shown in Figure 8 as Curves (a) through (d). This behavior can be attributed to the division of the 2D APL functions in Equation 7.

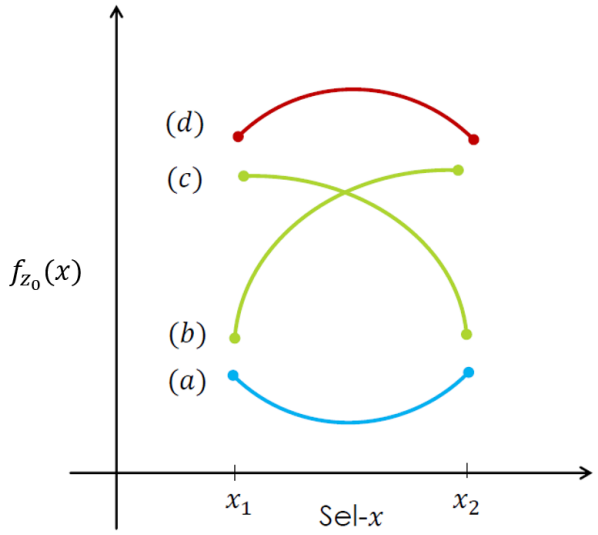


Figure 8: Behavior of the $f_z(x)$ function

Induced Slope Behaviour

Consider the (X, Z) *slice* that corresponds to a fixed value of Y. We now show that for any axis-parallel line segment within this slice, the maximum sub-optimality along the line occurs at one of its *end-points*. The lemma for line segments that are parallel to the X axis is given below (a similar statement applies to the Z axis).

Lemma 4.3 *Given a line segment $Z = z_o$ that is parallel to the X axis, the maximum suboptimality occurs at one of the end points ($(0, z_o)$ and $(1, z_o)$) if the slope $f'_{z_o}(x)$ is either*

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

Proof When the slope of $f'_{z_o}(x)$ is monotonically non-decreasing (i.e. Condition (1) is satisfied), the inflation function curve that connects the two points is guaranteed to lie below the straight line joining the two points – Curve (a) in Figure 8 shows an example of this situation. This ensures that the α along the given line segment is always less than or equal to the α at one of the end-points of the segment.

If, on the other hand, $f'_{z_o}(x)$ is monotonically decreasing, then the possible behaviors of the inflation function $f_{z_o}(x)$ are shown in curves (b) through (d) in Figure 8. Curves (b) and (c) denote the behavior of the inflation function when Condition (2) is satisfied, and clearly the value of the inflation function is below at least one end-point in the given range. Curve (d) however represents scenario where the local α_{local} does not occur at one of the end-points and hence does not satisfy either conditions of the lemma.

We now show that if the above behavior holds for two different values of Z, it gets induced for all *intermediate* Z values (a similar statement applies for line segments that are parallel to the Z axis).

Lemma 4.4 *If the slope of $f'_z(X)$ 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).*

Proof For fixed z, function f in Equation 7 reduces to

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

where

$$c_1 = (a_1 + a_3 z), c_2 = (a_2 z + a_4) d_1 = (b_1 + b_3 z), d_2 = (b_2 z + b_4) \tag{8}$$

and its slope is given by

$$f'_z(x) = \frac{c_1 d_2 - c_2 d_1}{(d_1 x + d_2)^2} \tag{9}$$

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 5, 6 and 8 we know that N can be written as the following function of z:

$$\begin{aligned} N(z) &= (a_1 + a_3 z)(b_2 z + b_4) - (a_2 z + a_4)(b_1 + b_3 z) \\ &= (a_3 b_2 - a_2 b_3) z^2 + (a_1 b_2 + a_3 b_4 - b_1 a_2 - a_4 b_3) z + a_1 b_4 - b_1 a_4 \\ &= l_1 z^2 + l_2 z + l_3 \end{aligned} \tag{10}$$

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 $f'_z(X)$ is decreasing $\forall z \in (z_1, z_2)$. Hence the Lemma.

Leveraging the above lemma, it immediately follows that if the slope behavior holds at the extreme Z selectivity values of 0 and 1, then the behavior is true throughout the entire range of Z. Further, similar lemmas apply to the X range as well. Putting together all this machinery leads to our main theorem:

Theorem 4.5 *(Perimeter Inflation) If any of the conditions C1 through C6 in Table 4 is satisfied, computing α_Y along the perimeters of its boundary walls is sufficient to establish α_Y within the entire* PSS.

Table 4: Sufficient Conditions for Perimeter Inflation

	Left Boundary	**Right Boundary**	**Top Boundary**	**Bottom Boundary**
C1	-	-	$f_1''(x) \geq 0$	$f_0''(x) \geq 0$
C2	$f_z'(0) \leq 0$	-	$f_1''(x) < 0$	$f_0''(x) < 0$
C3	-	$f_z'(1) \geq 0$	$f_1''(x) < 0$	$f_0''(x) < 0$
C4	$f_0''(z) \geq 0$	$f_1''(z) \geq 0$	-	-
C5	$f_0''(z) < 0$	$f_1''(z) < 0$	$f_x'(1) \geq 0$	-
C6	$f_0''(z) < 0$	$f_1''(z) < 0$	-	$f_x'(0) \leq 0$

Proof Consider the C1 condition in Table 4: Since $f_z''(x) \geq 0$ (i.e slope $f_z'(x)$ is non-decreasing) at the T-B boundaries, then from Lemma 4.4, we know that the slope $f_z'(x)$ is non-decreasing throughout the range (z_1, z_2).

Moving on to the C2 and C3 conditions: Since $f_z''(x) < 0$ (i.e slope $f_z'(x)$ is decreasing) at the T-B boundaries, then from Lemma 4.4, we know that the slope $f_z'(x)$ 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 $f_{z_o}'(x_1) \leq 0$ (C2) or $f_{z_o}'(x_2) \geq 0$ (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 α_Y occurs at one of the end-points of these lines as a result of the slope conditions given in Lemma 4.3 are satisfied. Since the union of all such line-segments is the given region, α_Y occurs at the perimeter. Similar arguments can be used to show that the perimeter is sufficient for finding the α_Y when conditions C4, C5 or C6 are satisfied.

Relaxing the APL assumption

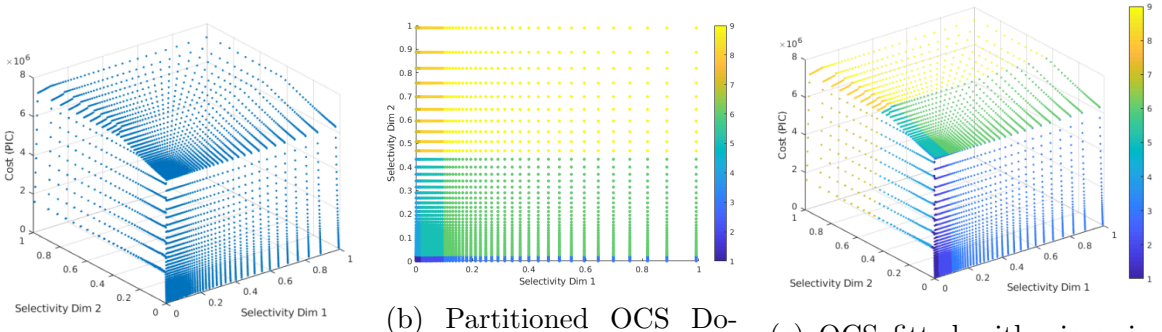
In the above analysis, we assumed that the OCS follows the *APL* property (Equation 4), but this may not always hold in practice. However, OCS typically exhibit *piece-wise* APL [11]. Moreover, our results require the piece-wise APL behavior to hold only for the 2D *slices* of the PSS, and not necessarily for the entire domain of the OCS as a whole.

The idea now is to divide the domain of the OCS into its constitutent APL-compliant pieces, and then apply the above-mentioned perimeter test independently to each of these pieces. Now if the perimeter of each of these regions satisfy any of the conditions from C1 to C6 of Table 4 then by Theorem 4.5 the α_Y occurs on the perimeter. But we first have to identify the pieces – for this purpose, we explicitly fit the OCS 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 OCS into non-overlapping regions. The perimeter test can then be used on these regions in isolation.

Moreover, an important point to note here is that we do not require an accurate *quantitative fit*, but only an accurate *qualitative fit* – that is, the slope behavior should be adequately captured.

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 9a. The 2D input domain of the OCS, 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,

$$f(x, y) = ax + by + cxy + d \tag{11}$$



(a) Original OCS

(b) Partitioned OCS Domain

(c) OCS fitted with piecewise functions

Figure 9: Original OCS and OCS fitted with an APL function per region of the partitioned input domain

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 9b. The computational complexity of the `MaxSelRemoval` perimeter algorithm for k regions is $O(2^{D-1} * D * res * k)$.

The problem now is identification of these regions where the Function 11 fits nicely. This is done using the K-subspace clustering methods for 2D and higher dimensional planes as described in [22] shown in Figure 9c. But using k-subspace clustering methods for identifying and fitting the planes we need to provide the OCS 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 OCSs we do the following, we first fit the 1D OCS with a piecewise linear function. This operation is done by using the K-subspace clustering [22] method for line shaped clusters shown in Figures 10a and 10b. 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 :

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

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

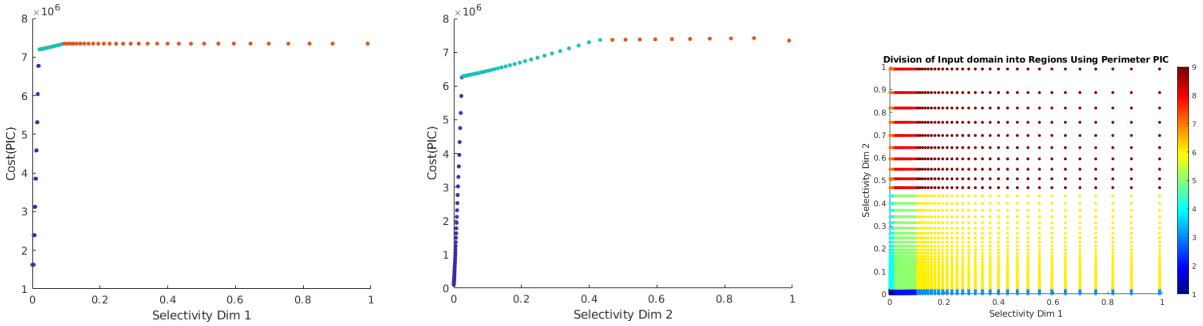
3. The objective function is to minimize the total distance (dispersion) of the cluster, which is $\min_{c_k, a_k} \sum_{i \in C_k} Dist(x_i, C_k)$.

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 10c which is almost identical to the domain decomposition in Figure 9b which was obtained using the PIC values for the entire region.

We then experimentally verify if the Function 11 fits well to the PIC restricted to the regions obtained from the combinations of the breakpoints of the piecewise linear functions at perimeter.

If the inflation function f satisfies anyone the conditions in Table 4 then the α occurs at the perimeter of one of these regions, since they together constitute the entire input domain.



(a) 1D OCS at perimeter Side 1

(b) 1D OCS at perimeter Side 2

(c) Dividing Input Domain using OCSs at perimeter

Figure 10: K-Subspace Clustering of 1D OCSs with linear clusters

Extension to Higher Dimensions

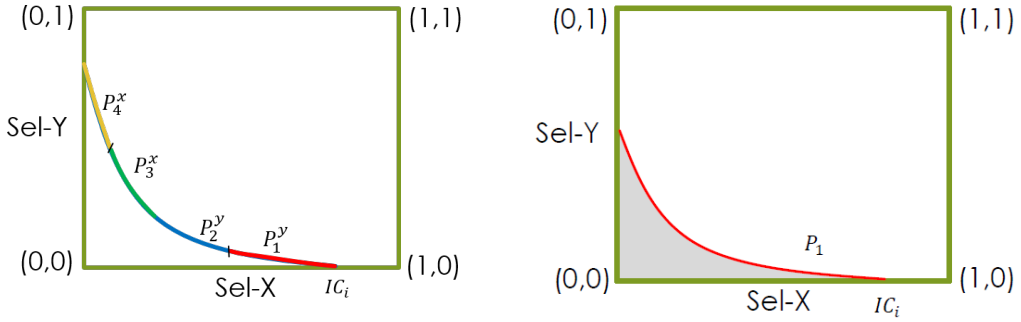
Our above analysis of the Perimeter Inflation procedure was carried out for a 3D `PSS`. For handling a higher dimension `PSS`, we simply need to consider all its 2D-faces. Specifically, if the perimeters of all the 2D-faces of the `PSS` satisfy any of the C1 to C6 conditions (Table 4), then we know that α_d occurs at the `PSS` perimeter, courtesy Theorem 4.5.

5 WeakDimRemoval techniques

The dimensions retained post `SchematicRemoval` and `MaxSelRemoval` techniques are the dimensions for which the `ESS` is constructed, that we assume to be D in number for the ease of notations. After the `ESS` construction the *iso-cost contours* are identified for `SB`'s execution. Here, an iso-cost contour, corresponding to cost C, represents the connected selectivity curve along which the cost of the optimal plan is C. A sample contour can be seen in Figure 11a shown as colored 1D curve. Series of contours starting from the lowest to highest cost in `ESS`, with geometric progression of two, are constructed. The key idea in `SB` is to perform D plan executions per contour from the lowest cost contour, until the actual selectivities of all the `epps` are explicitly learnt. Finally, the optimal plan is identified and executed for query completion. In this section we show how can we reduce the number of plan executions per-contour from D, to attain a tighter MSO. Again for ease of exposition, we consider the base case of $D = 2$, with `epp` X and Y. Then, move on to the 3D scenario to present the subtleties of the algorithm, from where it can easily be generalized to arbitrary dimensions.

5.1 `WeakDimRemoval` 2D scenario

We use the sample the 2D `ESS`, shown in Figure 11a, for ease of exposition of the algorithm. The figure depicts the iso-cost contour $\mathcal{IC}_i$, associated with cost $\texttt{CC}_\texttt{i}$, and annotated with the optimal plans P_1, P_2, P_3 and P_4. Note that the cost of these four plan on the contour locations costs $\texttt{CC}_\texttt{i}$. In `SB` each of these contour plans tries to individually and incrementally learn selectivities of the two `epp`s. `SB` carefully assign an `epp` for a contour plan to learn its selectivity, in order to achieve MSO guarantees. This mode of execution of plans trying to learn individual `epp` selectivities is referred to as *spill-mode execution of the plan while spilling on the* `epp`. For instance, in Figure 11a, the plan P_1 is annotated as P_1^y to indicate that it spills on the `epp` Y during execution.



(a) Contour on a 2D ESS

(b) `WeakDimRemoval` for $D = 2$

Contour Plan's Learning `epp` in `SB`

For each plan on the contour $\mathcal{IC}_i$, `SB` chooses an `epp` on which that plan needs to be *spilled*, so that the cost-budget for the plan is utilized to maximally learn the selectivity of that predicate only. This choice is based on the plan structure and the ordering of its constituent pipelines. We use this critical component of `SB` for our `WeakDimRemoval` technique, which is described next.

The 2D Algorithm

As mentioned before, `SB` requires at most two executions per contour until all the actual selectivity for both the `epp`s are learnt. Let us say that dimension X is removed (we discuss this choice later) using the 2D `WeakDimRemoval` algorithm. The idea in the algorithm, is to *piggyback* X's plan execution (and, hence its selectivity learning) along with Y-spilling plans. This is achieved by considering all plans that are X-spilling to be Y-spilling, in short, by ignoring the error-prone X-predicate in the pipeline order. Thus, we end up in all plans in a contour to be Y spilling. Finally, in order to make `WeakDimRemoval` MSO-efficient, we choose a plan on the contour which is relatively the cheapest on all locations of the contour, which is captured by an inflation factor. The maximum of these inflation factor's across all the contours is captured by β_X.

Now, if β_X is low such that $MSO_{SB}(1) * (1 + \beta_X) < MSO_{SB}(2)$, then `WeakDimRemoval` is successful for reduction in the MSO. In essence, we say that the *weak* dimension, X's execution is piggybacked by its *strong* dimension counterpart Y.

Algorithm Trace Let us now trace the above algorithm for our example `ESS` and contour $\mathcal{IC}_i$. First, we choose all the contour plans, which is P_1 to P_4. All these plans are assigned to spill on `epp` Y. Then, each of these four plans, are costed at all the contour locations, in order to find

the best one-plan replacement with the least inflation factor. In this scenario, P_1 happens to be our best replacement plan, as shown in Figure 11b.

Proof of Correctness

The algorithm's correctness, in order to achieve the desired MSO, is primarily dependent on the lower bound of selectivity learning of a plan while piggybacking the executions.

Lemma 5.1 (Piggybacked Execution) *Consider the contour plan P_r which replaces all the plans on contour $\mathcal{IC}_i$ with an cost inflation factor of β_X. Further, let P_r is assigned to spill on Y and executed with budget $\texttt{CC}_\texttt{i}(1+\beta_X)$. Then, then we either learn: (a) the exact selectivity of Y, or (b) infer that q_a lies beyond the contour.*

Proof $\mathcal{IC}_i$ represents the set of points in the ESS having their optimal cost equal to $\texttt{CC}_\texttt{i}$. The cost of all points $q \in \mathcal{IC}_i$ is at most $\texttt{CC}_\texttt{i}(1+\beta_X^i)$ when costed using P_r. Now when the plan P_r is executed in the spill-mode with cost budget $\texttt{CC}_\texttt{i}(1+\beta_X^i)$ 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 $q \in \mathcal{IC}_i$ with maximum sub-optimality of β_X^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 $\texttt{CC}_\texttt{i}(1+\beta_X^i)$ 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 $q.y$, $\forall q \in \mathcal{IC}_i$, since P_r is costed for all $q \in \mathcal{IC}_i$. Hence, q_a lies beyond the contour $\mathcal{IC}_i$.

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 $q \in \texttt{CC}_\texttt{i}$, all the selectivity combinations of $(q.x, q.y)$, $\forall q \in \mathcal{IC}_i$ get considered. Hence, for all these combinations the sum of the quantity $\sum_{N \in Upstream(N_Y)} N.cost +$ $N_Y.cost \leq \texttt{CC}_\texttt{i}(1+\beta_X^i)$. 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 $q \in \mathcal{IC}_i$ which implies that $Cost(P_{q_a}, q_a) > \texttt{CC}_\texttt{i}$ 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 β_X^i for each contour $\mathcal{IC}_i$ from $i = 1 \rightarrow m$. Let $\beta_X = \max_{i=1\rightarrow m} \beta_X^i$.

Lemma 5.2 *The MSO for the 2D scenario when contour plan replacement is done along a single dimension X is $4(1+\beta_X)$.*

Proof The query processing algorithm executes the best 1-plan replacement, P_i, for each contour $\mathcal{IC}_i$, starting from the least cost contour. Each execution of P_i is performed with an inflated budget of $\texttt{CC}_\texttt{i}(1+\beta_X)$. Since each contour now has only 1 plan with fixed inflated budget, using the 1D SB algorithm with inflated contour budgets it is easy to show that the MSOfor the 2D scenario post WeakDimRemoval is equal to $MSO_{SB}(1) * (1+\beta_X) = 4 * (1+\beta_X)$.

It is important to note that β_X - which denotes the worst case sub-optimality incurred for making plan replacements along the dimension X is a function of the dimension X itself.

Hence, By doing piggybacked executions of *'weak'* dimensions (dimensions with low α) along the *'strong'* dimensions (dimensions with high α), WeakDimRemoval makes the MSO a function of impactful dimensions only.

5.2 WeakDimRemoval 3D Scenario

In this sub-section we see how the WeakDimRemoval technique can be extended to the 3D scenario, consisting of dimensions X, Y and Z, where we wish to do WeakDimRemoval along dimension X. As in the 2D scenario, all the plans on the contour become either Y-spilling or Z-spilling by ignoring the epp X in the pipeline order. 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.

The main idea of the algorithm as stated earlier is to execute two plans (one for each strong dimension) and piggyback the execution of the weak dimension along with these strong ones. In our case, the execution of X is piggy backed with Y and Z. Let q^x_{sup}, q^x_{inf} denote the points having maximum and minimum X-selectivity on the contour respectively. Also, let $\mathsf{Sup}_x = q^x_{sup}.x$ and $\mathsf{Inf}_x = q^x_{inf}.x$. Let us first characterize the geometry of the contours based on the minimum and maximum selectivities of the replaced dimension X, captured by $X = \mathsf{Sup}_x$ and $X = \mathsf{Sup}_x$ respectively. There are three possibilities:

1. a 2D contour line on the $X = \mathsf{Inf}_x$ slice and a point on $X = \mathsf{Sup}_x$ slice
2. a 2D contour line on the $X = \mathsf{Sup}_x$ slice and a point on $X = \mathsf{Inf}_x$ slice
3. a 2D contour line on both $X = \mathsf{Inf}_x$ and $X = \mathsf{Sup}_x$ slices

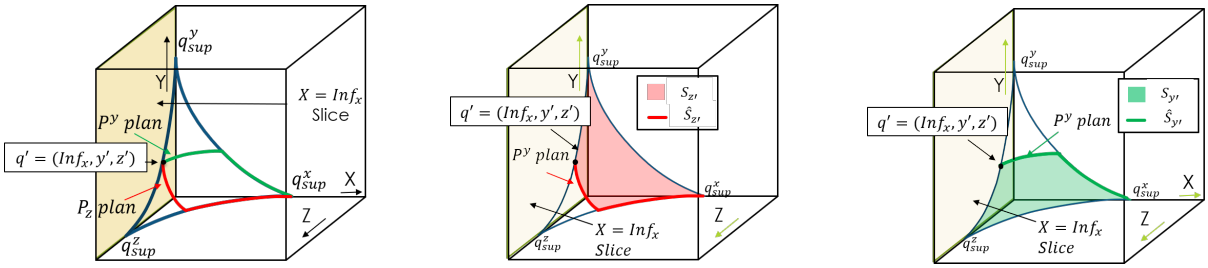
The rest of the section and figures correspond to the Case 1, but all the Lemmas and Theorems are easily generalizable for all the cases mentioned above.

To piggyback X's execution with Y, consider a point q' on the $X = \mathsf{Inf}_x$ slice, let its coordinates be such that $q' = (\mathsf{Inf}_x, y', z')$. This is shown in Figure 12a. Let us define the set $S_{y'} := \{q | q \in \mathcal{IC}_i \text{ and } q.y \leq y'\}$, that contains all the (x, y) selectivity combinations pertaining to the contour such that $y \leq y'$.

We now construct the minimal (x, y)-dominating set, that spatially dominates all points in $S_{y'}$ denoted by $\hat{S}_{y'}$. Formally,

$$\hat{S}_{y'} := \forall q \in S_{y'}, \exists \hat{q} \in \hat{S}_{y'} \text{ such that } (\hat{q}.x, \hat{q}.y) \succeq (q.x, q.y) \tag{12}$$

We do the best 1-plan replacement (where Y-spilling plans are chosen from P^T) for the set $\hat{S}_{y'}$, and say we get the plan P^y (shown in Figure 12c) . Now by PCM all the (x, y) selectivity combinations of the set $S_{y'}$ get covered by plan P^y since it is costed on the spatially dominating set of $S_{y'}$. Similarly, we can obtain a P^z plan for q' (shown in Figure 12b). The plan with minimum sub-optimality among the two, is chosen as the plan replacement for q' (shown in Figure 12a).



(a) Choice of a Plan for a point q'

(b) Choice of P^z Plan for a point q'

(c) Choice of P^y Plan for a point q'

Figure 12: `WeakDimRemoval` 3D Scenario Phase 1

For all such q' on the $X = \mathsf{Inf}_x$ slice (the 2D contour line on the slice), we either assign it a P^y or P^z plan i.e. a Y-spilling plan or a Z-spilling plan respectively. The point with maximum Y-coordinate which has a Y- spilling plan assigned to it is called as q^y_{max} and the plan corresponding to it as P^y_{max}. Similarly we obtain q^z_{max}, with the corresponding Z-spilling plan P^z_{max}. This procedure is depicted in Figure 13a where the green lines correspond to the points covered by Y-spilling plans and the red corresponding to the points covered by Z-spilling plans.

Let $\mathcal{Y} = q^y_{max}.y$ and $\mathcal{Z} = q^z_{max}.z$,

Lemma 5.3 *Let the following sets be defined as* $S_{\mathcal{Y}} := \{q | q \in \mathcal{IC}_i \text{ and } q.y \leq \mathcal{Y}\}$ *and* $S_{\mathcal{Z}} := \{q | q \in \mathcal{IC}_i \text{ and } q.z \leq \mathcal{Z}\}$. *Then every point* $q \in \mathcal{IC}_i$ *belongs to either* $S_{\mathcal{Y}}$ *or* $S_{\mathcal{Z}}$.

Proof Let us prove by contradiction. We know that q^y_{max} is the point on $X = \mathsf{Inf}_x$ slice which is covered by P^y_{max}, let its coordinates be $q^y_{max} := (\mathsf{Inf}_x, \mathcal{Y}, z)$. Analogously let $q^z_{max} := (\mathsf{Inf}_x, y, \mathcal{Z})$. It is evident that $q^y_{max}.z \leq \mathcal{Z}$, also $q^z_{max}.y \leq \mathcal{Y}$. Hence the point $\bar{q} := (\mathsf{Inf}_x, \mathcal{Y}, \mathcal{Z})$ 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 $\tilde{q} \notin S_{\mathcal{Y}}$ and $\tilde{q} \notin S_{\mathcal{Z}}$. This implies that $\tilde{q} \succ \bar{q}$ which further implies $cost(\tilde{q}) > cost(\hat{q}) \geq \mathtt{CC_i}$. Hence $\tilde{q} \notin IC_i$. Hence the proof.

Hence $(S_{\mathcal{Y}} \cup S_{\mathcal{Z}})$ covers all the points of the contour IC_i. This is depicted in Figure 13b, where the set $S_{\mathcal{Y}}$ covers the green area and the thick green curve is the set $\hat{S}_{\mathcal{Y}}$.

(a) Choosing the P^y_{max} and P^z_{max} plans

(b) Sets $S_{\mathcal{Y}}$, $S_{\mathcal{Z}}$, $\hat{S}_{\mathcal{Y}}$ and $\hat{S}_{\mathcal{Z}}$

Figure 13: `WeakDimRemoval` 3D Scenario Phase 2

Lemma 5.4 *If* P^y_{max} *plan is costed on the set* $\hat{S}_{\mathcal{Y}}$ *with* β^y_X *as maximum sub-optimality and if* P^y *is executed in spill-mode with budget* $(1 + \beta^y_X)\mathtt{CC_i}$ *and does not complete then* $q_a \notin S_{\mathcal{Y}}$.

Proof Since P^y_{max} plan is costed on the set $\hat{S}_{\mathcal{Y}}$ with β_y as maximum sub-optimality, and by definition of the set $\hat{S}_{\mathcal{Y}}$, the plan P^y_{max} essentially covers all the combinations of X and Y selectivity pairs for all the points q belonging to contour $\mathcal{IC}_i$ such that $q.y \leq \mathcal{Y}$. This is precisely the set $S_{\mathcal{Y}}$. Hence if the point $q_a \in S_{\mathcal{Y}}$ then the spill-mode execution with plan P^y_{max} completes (from Lemma 5.1) and thereby the lemma follows.

Similarly we can prove the following Lemma,

Lemma 5.5 *If P^z_{max} plan is costed on the set $\hat{S}_{\mathcal{Z}}$ with β^z_X as maximum sub-optimality and if P^z is executed in spill-mode with budget $(1+\beta^z_X)CC_i$ and does not complete then $q_a \notin S_{\mathcal{Z}}$.*

Corollary 5.6 *If the spill-mode executions of both the plans, P^y_{max} with budget $(1+\beta^y_X)\texttt{CC}_\texttt{i}$ and P^z_{max} with budget $(1+\beta^z_X)\texttt{CC}_\texttt{i}$, do not complete then q_a lies beyond the contour IC_i.*

Proof From Lemmas 5.4 and 5.5 we can infer that $q_a \notin S_{\mathcal{Y}}$ and $q_a \notin S_{\mathcal{Z}}$. This implies $q_a \notin (S_{\mathcal{Y}} \cup S_{\mathcal{Z}})$ and from Lemma 5.3 we can conclude that q_a lies beyond the contour $\mathcal{IC}_i$.

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 5.7 *In 3D-scenario the* `WeakDimRemoval` *Strategy ensures that at most two plans are executed from each of the contours $\mathcal{IC}_1, ..., \mathcal{IC}_{k+1}$, 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 $\mathcal{IC}_h$, where $1 \leq h \leq k+1$. We know that at most two plans are required to be executed in each of the contours $\mathcal{IC}_1, \ldots, \mathcal{IC}_h$ (from Lemma 5.6). 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 $\mathcal{IC}_h$, resulting in three plans being executed in $\mathcal{IC}_h$, and one plan each in contours $\mathcal{IC}_{h+1}$ through $\mathcal{IC}_{k+1}$.

Let $\beta^i_X = \max(\beta^y_X, \beta^z_X)$ for each of the contours from $i = 1 \rightarrow m$. Also let $\beta_X = \max\limits_{\{i=1\rightarrow m\}} \beta^i_X$.

We now analyze the worst-case cost incurred by `SB` after the Contour Plan Replacement strategy. For this, we assume that the contour with three plan executions is the costliest contour $\mathcal{IC}_{k+1}$. Since the ratio of costs between two consecutive contours is 2, the total cost incurred by `SB` is bounded as follows:

$$\begin{aligned}
TotalCost &\leq 2 * \texttt{CC}_\texttt{1}(1+\beta_X) + ... + 2 * \texttt{CC}_\texttt{k}(1+\beta_X) + 3 * \texttt{CC}_\texttt{k+1}(1+\beta_X) \\
&= (1+\beta_X)(2 * \texttt{CC}_\texttt{1} + .. + 2 * 2^{k-1}\texttt{CC}_\texttt{1} + 3 * 2^k\texttt{CC}_\texttt{1}) \\
&= (1+\beta_X)(2 * \texttt{CC}_\texttt{1}(1 + 2 + ...2^k) + 2^k * \texttt{CC}_\texttt{1}) \\
&= (1+\beta_X)(2 * \texttt{CC}_\texttt{1}(2^{k+1} - 1) + 2^k * \texttt{CC}_\texttt{1}) \\
&\leq (1+\beta_X)(2^{k+2} * \texttt{CC}_\texttt{1} + 2^k * \texttt{CC}_\texttt{1}) = (1+\beta_X) * 5 * 2^k * \texttt{CC}_\texttt{1}
\end{aligned}$$

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

$$MSO \leq \frac{(1+\beta_X) * 5 * 2^k * \texttt{CC}_\texttt{1}}{2^{k-1} * \texttt{CC}_\texttt{1}} = 10(1+\beta_X)$$

leading to the theorem:

Theorem 5.8 *With the Contour Plan Replacement done along dimension X, the MSO for the 3-D Scenario is* $10(1+\beta_X) = MSO_{\texttt{SB}}(2) * (1+\beta_X)$.

If $MSO_{\texttt{SB}}(2) * (1+\beta_X) < MSO_{\texttt{SB}}(3)$ then the dimension X is successfully replaced by `WeakDimRemoval` technique.

5.3 `WeakDimRemoval` Overheads

Let the maximum number of plans on any contour be denoted by ρ, the maximum number of contours be m. The `ESS` dimensionality post `SchematicRemoval` and `MaxSelRemoval` is $D - k_r$, which makes the size of the contour r^{D-k_r-1}. Then the effort required to do `WeakDimRemoval` is of the order $O(\rho * m * r^{D-k_r-1})$ Abstract Plan Costing (APC) calls. The APC calls are typically at least 100 times faster than usual optimizer calls, since the optimizer does not need to come up with a plan for the given location, instead it just needs to cost the *specified plan* using its cost model which makes it extremely economical. By doing an intra-contour anorexic plan reduction [8], we have $\rho_{anorexic}$: the maximum number of reduced plans on any contour, which is typically less than 10.

$$\begin{aligned}
\frac{Overheads((\texttt{PR}))}{Overheads(\texttt{SB})} &= \frac{(\rho_{anorexic} * m * r^{D-k_r-1}) * \text{APC calls}}{r^{D-k_r} * \text{OPT calls}} \\
&= \frac{(\rho_{anorexic} * m * r^{D-k_r-1}) * 10^{-2}}{m * r^{D-k_r-1}} \\
&= \frac{\rho_{anorexic} * m}{r * 100} \\
&\leq \frac{1}{r} (\text{For typical values of } \rho_{anorexic} \text{ and } m)
\end{aligned}$$

This makes the overheads of `WeakDimRemoval` just 1% of the overall required compile time effort for the resolution $r = 100$.

6 Experimental Evaluation

Having described the `DimRed` technique, we now turn our attention to its empirical evaluation. The experimental framework is described first, followed by an analysis of the results.

6.1 Database and System Framework

All our experiments were carried out on a generic HP Z440 multi-core workstation provisioned with 32 GB RAM, 512 GB SSD and 2TB HDD. The database engine was a modified version of the PostgreSQL 9.4 engine [20], with the primary additions being: (a) *Selectivity Injection*, required to generate the POSP overlay of the `PSS`; (b) *Abstract Plan Costing*, required to cost a specific plan at a particular `PSS` location; (c) *Abstract Plan Execution*, required to force the execution engine to execute a particular plan; (d) *Plan Recosting*, required to cost an abstract plan for a query; and (e) *Time-limited Execution*, required to implement the calibrated sequence of executions with associated time budgets.

Our test workload is comprised of a representative suite of complex OLAP queries, which are all based on queries appearing in the synthetic TPC-DS benchmark and the real-data JOB

benchmark [17].[4] Specifically, for TPC-DS, we have evaluated 31 SPJ queries operating on a database 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. On the other hand, for JOB, we have evaluated acyclic versions of 21 queries, operating on a database size of 5 GB, and also featuring a wide spectrum of join-graph geometries. More importantly, JOB features complex filter predicates involving string comparisons with the `LIKE` operator, and multiple predicates on a single base relation.

We also deliberately created challenging environments for `DimRed` by maximizing the range of cost values in the `PSS`, making it more likely for dimensions to be retained. This was achieved through an index-rich physical schema that created indexes on all the attribute columns appearing in the queries.

6.2 Goodness of OCS Fit (Surface and Perimeter)

For verifying the modelling of OCS described in Section 4, we consider the 2D faces of the `PSS` post `SchematicRemoval`, and divide the input domain first using the entire OCS surface and then using just the perimeter using the methods described in Section 4.4. Next, to measure the goodness of the fit we compute the *Normalized RMSE* and *Normalized Max Error*. The results in the Tables 5 and 6 show that we are able to achieve modest fitting with perimeter as well and hence validate our claims.

Table 5: RMSE (TPC-DS)

Query Number	Surface Fit		Perimeter Fit	
	Normalized RMSE	*Normalized Max Error*	*Normalized RMSE*	*Normalized Max Error*
Q03	11.60	24.51	14.68	25.99
Q07	15.79	22.16	17.30	25.61
Q12	16.50	16.56	23.00	25.09
Q15	10.62	16.63	11.82	17.05
Q18	10.48	21.79	16.63	19.94
Q19	16.45	22.68	17.36	26.83
Q21	16.34	27.32	17.02	28.71
Q22	11.33	14.84	14.48	15.53
Q26	11.42	22.02	11.61	24.58
Q27	16.94	27.32	17.32	28.65
Q29	15.20	16.33	19.29	20.57
Q36	8.47	23.15	15.36	28.94
Q37	9.58	23.29	12.06	20.23
Q40	15.48	21.48	17.12	29.80
Q42	9.09	26.96	17.56	28.04
Q43	14.99	22.42	16.19	24.25
Q52	15.14	21.70	18.14	29.30
Q53	14.39	21.32	17.25	27.72
Q55	9.03	14.72	18.47	27.11
Q62	15.78	22.78	16.40	34.14
Q63	9.90	10.28	11.78	19.50
Q67	10.19	11.21	10.44	12.29
Q73	14.52	19.30	16.36	24.55
Q82	10.56	17.36	10.84	25.43
Q84	16.78	27.29	17.03	29.51
Q86	15.35	19.54	17.28	23.29
Q89	10.59	19.92	19.37	29.49
Q91	16.42	22.22	13.88	24.53
Q96	14.17	27.84	20.07	29.89
Q98	14.20	15.70	15.02	26.86
Q99	14.88	15.63	18.30	29.87

Table 6: RMSE (JOB)

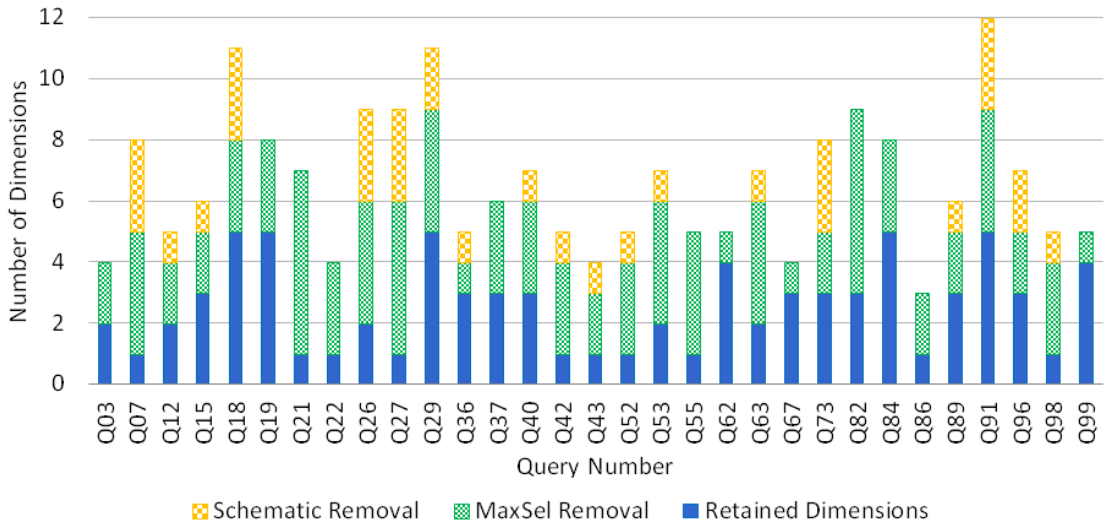
Query Number	Surface Fit		Perimeter Fit	
	Normalized RMSE	*Normalized Max Error*	*Normalized RMSE*	*Normalized Max Error*
Q01	15.30	20.62	18.02	26.58
Q08	11.86	15.82	18.81	29.15
Q09	12.40	13.65	14.95	19.51
Q10	14.31	22.79	16.06	24.59
Q11	12.27	19.12	15.11	21.35
Q12	11.18	16.13	15.69	19.70
Q13	7.91	10.14	18.64	27.85
Q14	13.06	13.54	15.86	25.83
Q15	15.43	16.74	18.62	20.39
Q16	16.77	25.68	19.13	16.46
Q19	15.11	24.27	15.41	26.93
Q20	10.19	16.54	18.76	27.40
Q21	9.84	19.68	12.74	26.69
Q22	8.77	19.29	15.99	23.52
Q23	12.91	17.63	16.92	20.62
Q24	11.36	17.39	13.95	19.39
Q25	11.72	11.83	14.90	16.50
Q26	9.23	14.11	17.35	25.95
Q28	12.46	17.19	19.06	26.32
Q29	9.11	11.88	13.23	21.01
Q33	11.83	20.65	13.83	25.32

[4]While the `DimRed` approach is conceptually applicable to the native queries in these benchmarks, the modifications are an artifact of our limited implementation of robustness-related features (e.g. selectivity injection) in the PostgreSQL engine.

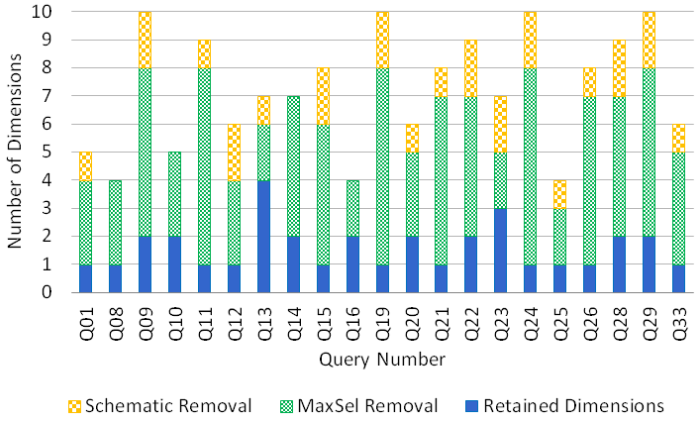
6.3 Validation of Perimeter Inflation

As discussed in Section 4, an efficient implementation of `MaxSelRemoval` requires the α_d corresponding to any dimension d to be located on the *perimeter* of either the `PSS` itself, or its piece-wise approximation. We have conducted a detailed validation of this behavioral assumption. Specifically, we carried out an offline exhaustive construction of the complete `PSS` and calculated the α_d for each dimension d. These values were then compared with the αs obtained by restricting the calculation to only the perimeter(s) of the `PSS`. The results showed that for *all* the queries in our workload, α did occur on the perimeter. In fact, for most of the dimensions in these queries, the α occurred not just on the perimeter, but at the *vertices* of the `PSS`.

6.4 Overheads Minimization Objective



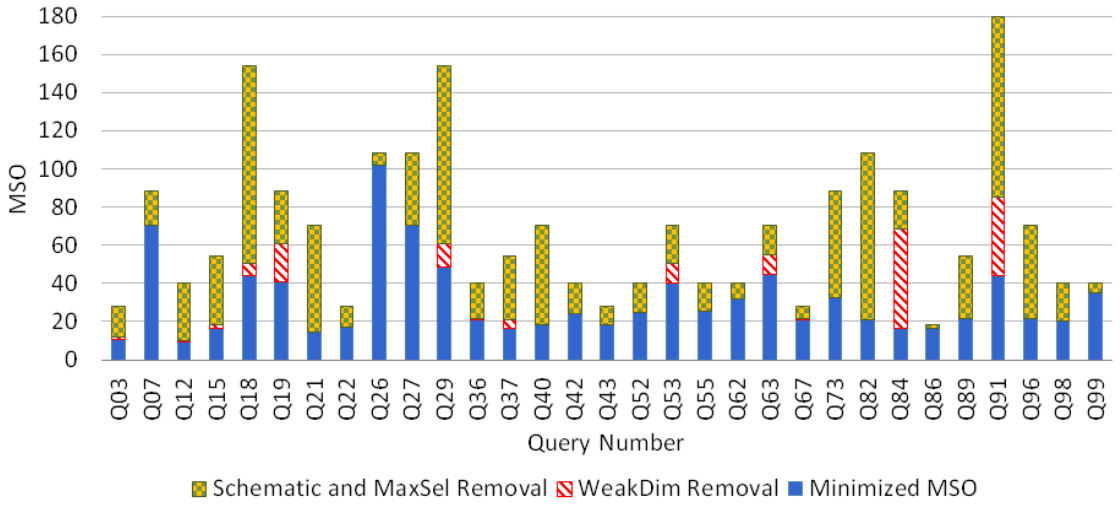
(a) TPC-DS Queries



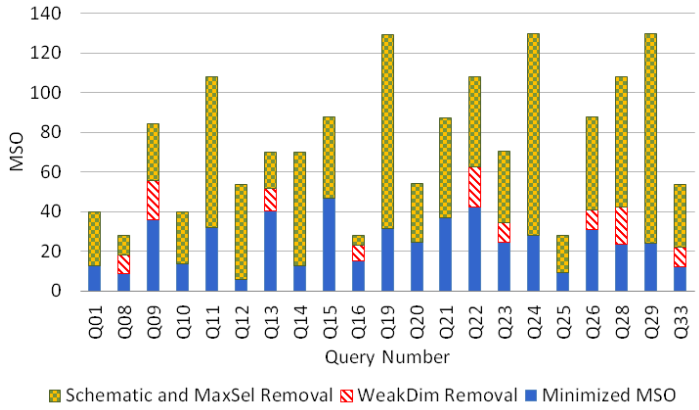
(b) JOB Queries

Figure 14: Dimensionality Reduction for Overheads Minimization

We now turn our attention to the `DimRed` performance on the overheads minimization metric, where the objective is to minimize the `PSS` dimensionality while retaining MSO-safety. The performance results for this scenario are shown in Figures 14a and 14b for the TPC-DS and JOB query suites, respectively. In these figures, the full height of each vertical bar shows the original



(a) TPC-DS Queries



(b) JOB Queries

Figure 15: MSO Profile for Overheads Minimization

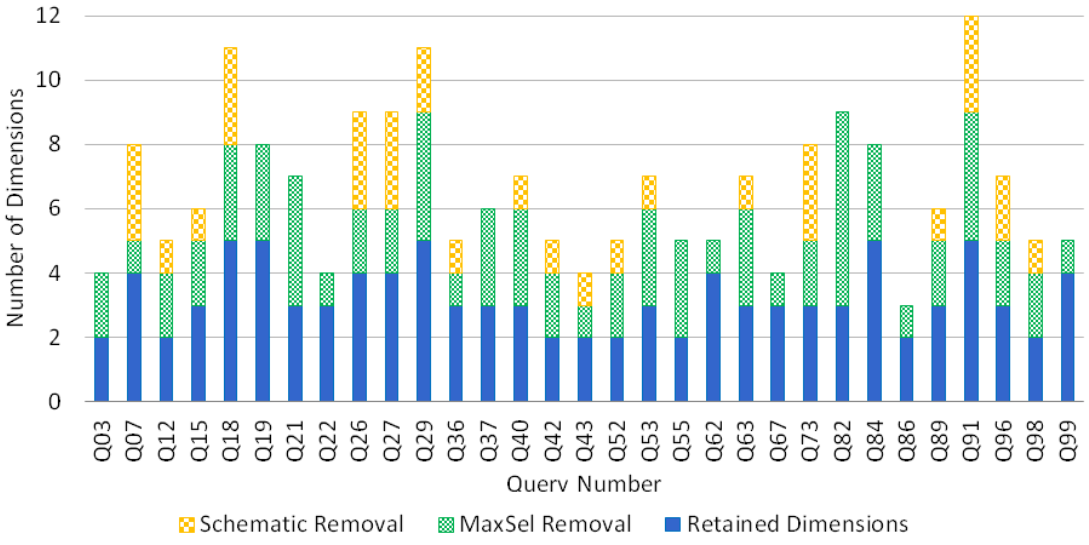
PSS dimensionality, while the bottom segment (blue fill) within the bar indicates the final ESS dimensionality, after reduction by the SchematicRemoval (yellow checks) and MaxSelRemoval (green braid) modules.

The important observation here is that across all the queries, the ESS dimensionality is essentially *"anorexic"*, being always brought down to five or less. In fact, for as many as 10 queries in TPC-DS and 11 queries in JOB the number of dimensions retained is just **1**! We also see that MaxSelRemoval usually plays the primary role, and SchematicRemoval the secondary role, in realizing these anorexic dimensionalities. Inspection of the retained dimensions showed that all the base filter predicates are removed from the PSS either by SchematicRemoval or by MaxSelRemoval, leaving behind only the high-impact join dimensions. Another observation is that SchematicRemoval removal is not as successful on the JOB benchmark as on TPC-DS – this is due to the complex filter predicates on the base relations. But by using the bounds provided by SchematicRemoval, MaxSelRemoval is successfully able to remove all of them with only a small MSO inflation. These results also justify our creation of an automated pipeline to replace the handpicking of dropped dimensions in the earlier literature.

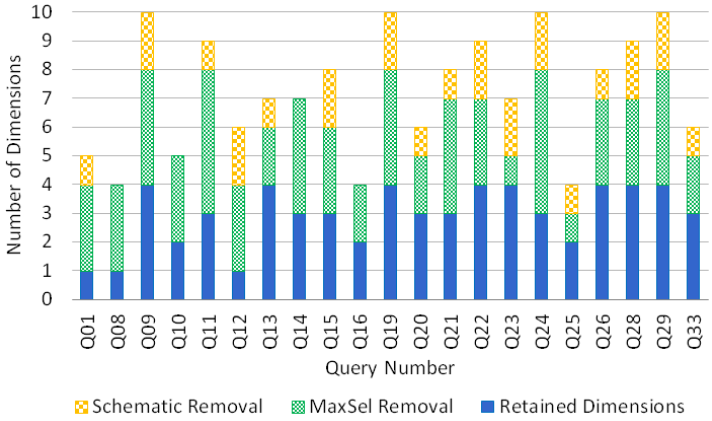
After the above dimensionality reduction, the next step in the DimRed pipeline is to try and improve the MSO through invocation of the WeakDimRemoval module. The resulting MSO val-

ues are shown in Figures 15a and 15b for the TPC-DS and JOB query suites, respectively. In these figures, the full height of each vertical bar shows the MSO of the original `PSS`, while the bottom segment (blue fill) within the bar indicates the final MSO, after initial improvements by `SchematicRemoval` and `MaxSelRemoval` (yellow-green checks) and subsequently by `WeakDimRemoval` (red lines). The important observation here is that for a majority of the queries, the final MSO is substantially lower than the starting value. For instance, with TPC-DS Q91, the MSO is tightened from 180 to 40, and with JOB Q19, the improvement is from 130 to 35. Overall, we find an average decrease of 50% and 67% for TPC-DS and JOB, respectively.

6.5 MSO Minimization Objective



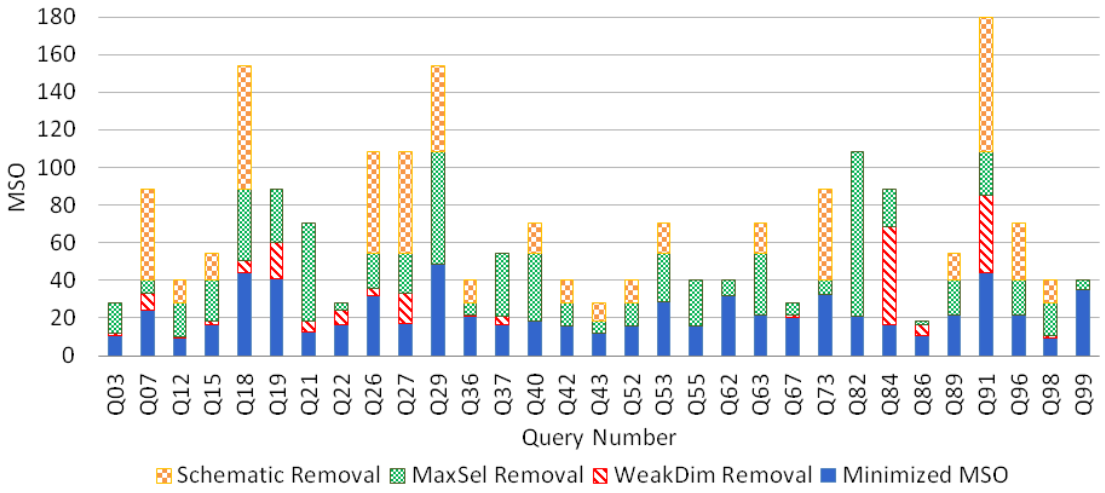
(a) TPC-DS Queries



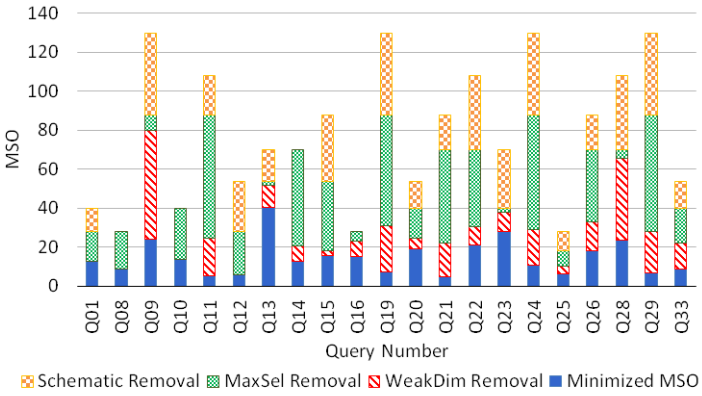
(b) JOB Queries

Figure 16: Dimensionality Reduction for MSO Minimization

We now turn our attention to the goal of dimensionality reduction with the objective of minimizing MSO, subject to the safety requirement. The dimensionality results for this alternative scenario are shown in Figures 16a and 16b for the TPC-DS and JOB query suites, respectively. In these figures, the full height of each vertical bar shows the original `PSS` dimensionality, while



(a) TPC-DS Queries



(b) JOB Queries

Figure 17: MSO Profile for MSO Minimization

the bottom blue segment within the bar indicates the final `ESS` dimensionality, after reduction by the `SchematicRemoval` (yellow checks) and `MaxSelRemoval` (green braids) modules.

As should be expected, the number of dimensions retained are slightly higher with MSO minimization as compared to overheads minimization. However, all queries still have less than or equal to five dimensions.

The corresponding MSO profile is shown in Figures 17a and 17b for the TPC-DS and JOB query suites, respectively confirm this claim. Again, the full vertical height captures the original MSO, and the bottom segment (blue fill) shows the final MSO, after improvements due to `SchematicRemoval` (yellow checks), `MaxSelRemoval` (green braid) and `WeakDimRemoval` (red lines).

In summary, for the TPC-DS queries, we obtain an average improvement of 62%, whereas for the JOB queries it is 77%.

6.6 Time Efficiency of `DimRed`

A plausible concern about `DimRed` is whether the overheads saved due to dimensionality reduction may be negated by the computational overheads of the pipeline itself. To address this issue, we

present in Table 7, a sample profile of DimRed's efficiency, corresponding to TPC-DS Query 91, which is the highest dimensionality query in our workload, featuring 6 filter and 6 join predicates. In the table, the optimizer calls made by the pipeline, and the overall time expended in this process, are enumerated. We find that the entire pipeline completes in less than 15 minutes, *inclusive* of the POSP overlay on the ESS, whereas the compilation efforts on the original PSS would have taken more than a year!

Table 7: DimRed Efficiency: TPC-DS Query 91

	Dimensionality	**MSO**	**Overheads (Opt Calls)**	Time (Secs)
PSS	12	180	4 quadrillion(10^{15})	¿ 1 year
Schematic Removal	9	108	$5 * 10^5$(MaxSel) + $32 * 10^5$(ESS) + $1.6 * 10^5$(WeakDim) = $38.6 * 10^5$	100 (MaxSel) + 640 (ESS) + 32 (WeakDim) ≈ 13 minutes
MaxSel Removal	5	84		
WeakDim Removal	2	44		

Table 8: DimRed Time Efficiency: TPC-DS (Overheads Minimization)

Query Number	**Overheads (Optimizer Calls)**				**Overheads(Time in Secs)**			
	MaxSel Calls	*ESS Calls*	*WeakDim Calls*	*Total*	*MaxSel*	*ESS*	*WeakDim*	*Total*
Q03	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q07	1600	20	1	1621	0.32	0.004	0.0002	**0.3242**
Q12	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q15	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q18	20480	3200000	160000	3380480	4.096	640	32	**676.096**
Q19	20480	3200000	160000	3380480	4.096	640	32	**676.096**
Q21	8960	20	1	8981	1.792	0.004	0.0002	**1.7962**
Q22	640	20	1	661	0.128	0.004	0.0002	**0.1322**
Q26	3840	400	20	4260	0.768	0.08	0.004	**0.852**
Q27	46080	20	1	46101	9.216	0.004	0.0002	**9.2202**
Q29	46080	3200000	160000	3406080	9.216	640	32	**681.216**
Q36	640	8000	400	9040	0.128	1.6	0.08	**1.808**
Q37	3840	8000	400	12240	0.768	1.6	0.08	**2.448**
Q40	3840	8000	400	12240	0.768	1.6	0.08	**2.448**
Q42	640	20	1	661	0.128	0.004	0.0002	**0.1322**
Q43	240	20	1	261	0.048	0.004	0.0002	**0.0522**
Q52	640	20	1	661	0.128	0.004	0.0002	**0.1322**
Q53	3840	400	20	4260	0.768	0.08	0.004	**0.852**
Q55	1600	20	1	1621	0.32	0.004	0.0002	**0.3242**
Q62	1600	160000	8000	169600	0.32	32	1.6	**33.92**
Q63	3840	400	20	4260	0.768	0.08	0.004	**0.852**
Q67	640	8000	400	9040	0.128	1.6	0.08	**1.808**
Q73	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q82	46080	8000	400	54480	9.216	1.6	0.08	**10.896**
Q84	20480	3200000	160000	3380480	4.096	640	32	**676.096**
Q86	240	20	1	261	0.048	0.004	0.0002	**0.0522**
Q89	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q91	491520	3200000	160000	3851520	98.304	640	32	**770.304**
Q96	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q98	640	20	1	661	0.128	0.004	0.0002	**0.1322**
Q99	1600	160000	8000	169600	0.32	32	1.6	**33.92**

Table 9: DimRed Time Efficiency: JOB (Overheads Minimization)

Query Number	Overheads (Optimizer Calls)				Overheads(Time in Secs)			
	MaxSel Calls	*ESS Calls*	*WeakDim Calls*	*Total*	*MaxSel*	*ESS*	*WeakDim*	*Total*
Q01	640	20	1	661	0.1280	0.0040	0.0002	**0.1322**
Q08	640	20	1	661	0.1280	0.0040	0.0002	**0.1322**
Q09	20480	400	20	20900	4.0960	0.0800	0.0040	**4.1800**
Q10	1600	400	20	2020	0.3200	0.0800	0.0040	**0.4040**
Q11	20480	20	1	20501	4.0960	0.0040	0.0002	**4.1002**
Q12	640	20	1	661	0.1280	0.0040	0.0002	**0.1322**
Q13	3840	160000	8000	171840	0.7680	32.0000	1.6000	**34.3680**
Q14	8960	400	20	9380	1.7920	0.0800	0.0040	**1.8760**
Q15	3840	20	1	3861	0.7680	0.0040	0.0002	**0.7722**
Q16	640	400	20	1060	0.1280	0.0800	0.0040	**0.2120**
Q19	20480	20	1	20501	4.0960	0.0040	0.0002	**4.1002**
Q20	1600	400	20	2020	0.3200	0.0800	0.0040	**0.4040**
Q21	8960	20	1	8981	1.7920	0.0040	0.0002	**1.7962**
Q22	8960	400	20	9380	1.7920	0.0800	0.0040	**1.8760**
Q23	1600	8000	400	10000	0.3200	1.6000	0.0800	**2.0000**
Q24	20480	20	1	20501	4.0960	0.0040	0.0002	**4.1002**
Q25	240	20	1	261	0.0480	0.0040	0.0002	**0.0522**
Q26	8960	20	1	8981	1.7920	0.0040	0.0002	**1.7962**
Q28	8960	400	20	9380	1.7920	0.0800	0.0040	**1.8760**
Q29	20480	400	20	20900	4.0960	0.0800	0.0040	**4.1800**
Q33	1600	20	1	1621	0.3200	0.0040	0.0002	**0.3242**

Table 10: DimRed Time Efficiency: TPC-DS (MSO Minimization)

Query Number	Overheads (Optimizer Calls)				Overheads(Time in Secs)			
	MaxSel Calls	*ESS Calls*	*WeakDim Calls*	*Total*	*MaxSel*	*ESS*	*WeakDim*	*Total*
Q03	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q07	1600	160000	8000	169600	0.32	32	1.6	**33.92**
Q12	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q15	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q18	20480	3200000	160000	3380480	4.096	640	32	**676.096**
Q19	20480	3200000	160000	3380480	4.096	640	32	**676.096**
Q21	8960	8000	400	17360	1.792	1.6	0.08	**3.472**
Q22	640	8000	400	9040	0.128	1.6	0.08	**1.808**
Q26	3840	160000	8000	171840	0.768	32	1.6	**34.368**
Q27	46080	160000	8000	214080	9.216	32	1.6	**42.816**
Q29	46080	3200000	160000	3406080	9.216	640	32	**681.216**
Q36	640	8000	400	9040	0.128	1.6	0.08	**1.808**
Q37	3840	8000	400	12240	0.768	1.6	0.08	**2.448**
Q40	3840	8000	400	12240	0.768	1.6	0.08	**2.448**
Q42	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q43	240	400	20	660	0.048	0.08	0.004	**0.132**
Q52	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q53	3840	8000	400	12240	0.768	1.6	0.08	**2.448**
Q55	1600	400	20	2020	0.32	0.08	0.004	**0.404**
Q62	1600	160000	8000	169600	0.32	32	1.6	**33.92**
Q63	3840	8000	400	12240	0.768	1.6	0.08	**2.448**
Q67	640	8000	400	9040	0.128	1.6	0.08	**1.808**
Q73	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q82	46080	8000	400	54480	9.216	1.6	0.08	**10.896**
Q84	20480	3200000	160000	3380480	4.096	640	32	**676.096**
Q86	240	400	20	660	0.048	0.08	0.004	**0.132**
Q89	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q91	491520	3200000	160000	3851520	98.304	640	32	**770.304**
Q96	1600	8000	400	10000	0.32	1.6	0.08	**2**
Q98	640	400	20	1060	0.128	0.08	0.004	**0.212**
Q99	1600	160000	8000	169600	0.32	32	1.6	**33.92**

Notwithstanding the above, it is still possible that there may be queries for which DimRed's overheads may prove to be impractically large. But this situation can also be addressed by leveraging the recently developed FrugalSpillBound algorithm [13], which provides a very attractive tradeoff between POSP overlay overheads and the MSO guarantee – specifically, an exponential decrease in overheads at the cost of a linear relaxation in MSO. Since our MSO improvements due to the pipeline are typically quite substantial, as highlighted in the above experiments, it is quite likely that even after the relaxation is applied, MSO-safety will continue to be retained. In fact, the attractive results presented in [13] were achieved in conjunction with the SchematicRemoval and MaxSelRemoval modules of DimRed.

Table 11: DimRed Time Efficiency: JOB (MSO Minimization)

Query Number	Overheads (Optimizer Calls)				Overheads(Time in Secs)			
	MaxSel Calls	*ESS Calls*	*WeakDim Calls*	*Total*	*MaxSel*	*ESS*	*WeakDim*	*Total*
Q01	640	20	1	661	0.1280	0.0040	0.0002	**0.1322**
Q08	640	20	1	661	0.1280	0.0040	0.0002	**0.1322**
Q09	20480	160000	8000	188480	4.0960	32.0000	1.6000	**37.6960**
Q10	1600	400	20	2020	0.3200	0.0800	0.0040	**0.4040**
Q11	20480	8000	400	28880	4.0960	1.6000	0.0800	**5.7760**
Q12	640	20	1	661	0.1280	0.0040	0.0002	**0.1322**
Q13	3840	160000	8000	171840	0.7680	32.0000	1.6000	**34.3680**
Q14	8960	8000	400	17360	1.7920	1.6000	0.0800	**3.4720**
Q15	3840	8000	400	12240	0.7680	1.6000	0.0800	**2.4480**
Q16	640	400	20	1060	0.1280	0.0800	0.0040	**0.2120**
Q19	20480	160000	8000	188480	4.0960	32.0000	1.6000	**37.6960**
Q20	1600	8000	400	10000	0.3200	1.6000	0.0800	**2.0000**
Q21	8960	8000	400	17360	1.7920	1.6000	0.0800	**3.4720**
Q22	8960	160000	8000	176960	1.7920	32.0000	1.6000	**35.3920**
Q23	1600	160000	8000	169600	0.3200	32.0000	1.6000	**33.9200**
Q24	20480	8000	400	28880	4.0960	1.6000	0.0800	**5.7760**
Q25	240	400	20	660	0.0480	0.0800	0.0040	**0.1320**
Q26	8960	160000	8000	176960	1.7920	32.0000	1.6000	**35.3920**
Q28	8960	160000	8000	176960	1.7920	32.0000	1.6000	**35.3920**
Q29	20480	160000	8000	188480	4.0960	32.0000	1.6000	**37.6960**
Q33	1600	8000	400	10000	0.3200	1.6000	0.0800	**2.0000**

7 Conclusions

The PlanBouquet and SpillBound algorithms have brought welcome robustness guarantees to database query processing. However, they are currently practical only for low-dimensional selectivity spaces since their compilation overheads are exponential in the dimensionality, and their performance bounds are quadratic in the dimensionality. In this paper, we have shown how seemingly high-dimensional queries can be systematically reduced to low-dimension equivalents without sacrificing the performance guarantees, thereby significantly increasing the coverage of the robust techniques.

We presented the DimRed pipeline, which leverages schematic, geometric and piggybacking techniques to reduce even queries with double digit dimensionalities to five or less dimensions. In fact, for quite a number of queries, the dimensionality came down to the lowest possible value of 1! Gratifyingly, not only could we dramatically decrease the overheads due to such reductions, but could also significantly improve the quality of the performance guarantee. Therefore, in an overall sense, DimRed offers a substantive step forward in making robust query processing feasible on current environments. Our techniques have been tested thus far on acyclic SPJ queries. In our future work, we would like to extend our implementation and analysis to both nested and cyclic queries.

References

[1] Dagstuhl Seminar. Robust Query Processing. www.dagstuhl.de/en/program/calendar/semhp/?semnr=10381, 2010.

[2] Dagstuhl Seminar. Robust Query Processing. www.dagstuhl.de/en/program/calendar/semhp/?semnr=12321, 2012.

[3] Dagstuhl Seminar. Robust Query Processing. www.dagstuhl.de/en/program/calendar/semhp/?semnr=17222, 2017.

[4] Surajit Chaudhuri, Venkatesh Ganti, and Luis Gravano. Selectivity estimation for string predicates: Overcoming the underestimation problem. In *IEEE ICDE Conf.*, 2004.

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

[6] A. Dutt and J. Haritsa. Plan bouquets: A fragrant approach to robust query processing. *ACM TODS*, 41(11), 2016.

[7] G. Graefe. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.

[8] D. Harish, P. Darera, and J. Haritsa. On the production of anorexic plan diagrams. In *VLDB Conf.*, 2007.

[9] D. Harish, P. Darera, and J. Haritsa. Identifying robust plans through plan diagram reduction. *PVLDB*, 1(1):1124–1140, 2008.

[10] H. Harmouch and F. Naumann. Cardinality estimation: An experimental survey. *PVLDB*, 11(4):499–512, 2017.

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

[12] S. Karthik, J. Haritsa, S. Kenkre, and V. Pandit. Platform-independent robust query processing. In *IEEE ICDE Conf.*, 2016.

[13] S. Karthik, J. Haritsa, S. Kenkre, and V. Pandit. A concave path to low-overhead robust query processing. *PVLDB*, 11(13):2183–2195, 2018.

[14] S. Karthik, J. Haritsa, S. Kenkre, V. Pandit, and L. Krishnan. Platform-independent robust query processing. *IEEE TKDE*, 2017.

[15] Hongrae Lee, Raymond T. Ng, and Kyuseok Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *VLDB Conf.*, 2007.

[16] Hongrae Lee, Raymond T. Ng, and Kyuseok Shim. Approximate substring selectivity estimation. In *EDBT Conf.*, 2009.

[17] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.

[18] Guy Lohman. Is query optimization a solved problem? `http://wp.sigmod.org/?p=1075`, 2014.

[19] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. Robust query processing through progressive optimization. In *ACM SIGMOD Conf.*, 2004.

[20] PostgreSQL. `http://www.postgresql.org/docs/9.4/static/release.html`.

[21] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access path selection in a relational database management system. In *ACM SIGMOD Conf.*, 1979.

[22] D. Wang, C. Ding, and T. Li. K-subspace clustering. In *ECML-PKDD Conf.*, 2009.