A Concave Path to Low-overhead Robust Query Processing

Srinivas Karthik Jayant Haritsa Sreyash Kenkre[1] Vinayaka Pandit[1]

Technical Report
TR-2018-01

Database Systems Lab
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560012, India

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

[1]IBM Research, India

Abstract

To address the classical selectivity estimation problem in database systems, a radically different query processing technique called `PlanBouquet` was proposed in [4]. In this approach, the estimation process is completely *abandoned* and replaced instead with a calibrated selectivity *discovery* mechanism. The beneficial outcome is that provable guarantees are obtained on worst-case execution performance, thereby facilitating robust query processing. An improved version of `PlanBouquet`, called `SpillBound` (`SB`), which significantly accelerates the selectivity discovery process, and provides platform-independent performance guarantees, was recently presented in [10].

A major limitation of `SpillBound`, however, is that its guarantees are predicated on expending enormous pre-processing efforts during query compilation, making it suitable only for canned queries that are invoked repeatedly. In this paper, we address this limitation by leveraging the fact that plan cost functions typically exhibit *concave down behavior* with regard to predicate selectivities. Specifically, we design `FrugalSpillBound`, which provably achieves extremely attractive tradeoffs between the performance guarantees and the compilation overheads. For instance, relaxing the performance guarantee by a factor of two typically results in at least *two orders of magnitude* reduction in the overheads. Further, when empirically evaluated on benchmark OLAP queries, the decrease in overheads is even greater, often reaching *three* orders of magnitude. Therefore, in an overall sense, `FrugalSpillBound` represents a substantive step forward with regard to achieving robust query processing for ad-hoc queries.

1 Introduction

The traditional approaches for optimizing declarative OLAP queries (e.g. [16, 6]) are contingent on estimating a host of *predicate selectivities* in order to find the optimal execution plan. For instance, even for the relatively simple TPC-H query shown in Figure 1, which lists the order dates for cheap parts, as many as *four* selectivities have to be estimated, corresponding to the filter predicate ($p\_retailprice < 1000$), two join predicates ($part \bowtie lineitem, orders \bowtie lineitem$), and the projection predicate ($o\_orderdate$).

```
select distinct o_orderdate 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 selectivity estimates are often significantly in error with respect to the actual selectivities encountered during query execution – to the extent that *orders of magnitude* errors have been routinely reported in the literature [1, 11, 12]. These estimation errors cumulatively result in highly sub-optimal choices of execution plans, and corresponding blowups in query response times. For instance, when Query 19 of the TPC-DS benchmark is executed on contemporary database engines, the worst-case slowdown, relative to a hypothetical oracle that magically knows the correct selectivities, can exceed a *million*! [4]

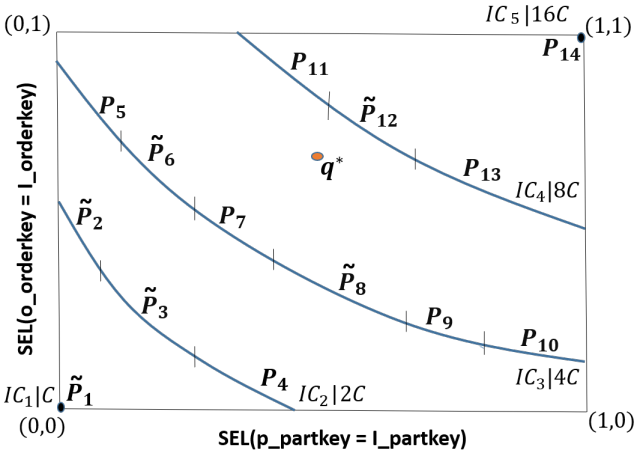


Figure 2: SpillBound Execution on 2D ESS

SpillBound To address the above chronic problem, a radically different query processing technique, called PlanBouquet, was proposed in [4]. In this approach, the highly brittle selectivity estimation process is completely *abandoned*, and replaced instead with a calibrated *discovery* mechanism. The beneficial outcome of the new construction is that *provable guarantees* are obtained on worst-case performance, thereby facilitating robust query processing. An improved version of PlanBouquet, called SpillBound (SB), which significantly accelerates the selectivity discovery process, and provides platform-independent performance guarantees, was recently presented in [10].

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

On this ESS space, a series of *isocost* contours, $\mathcal{IC}_1$ through $\mathcal{IC}_m$, are drawn – each isocost contour $\mathcal{IC}_i$ has an associated optimizer estimated cost $\texttt{CC}_\texttt{i}$, and represents the connected selectivity curve along which the cost of the optimal plan is equal to $\texttt{CC}_\texttt{i}$. Further, the contours are selected such that the cost of the first contour $\mathcal{IC}_1$ corresponds to the minimum query cost C at the origin of the space, and the cost of each of the following contours is *double* that of the previous contour. Therefore, in Figure 2, there are five hyperbolic contours, $\mathcal{IC}_1$ through $\mathcal{IC}_5$, with their costs ranging from $\texttt{CC}_1 = C$ to $\texttt{CC}_5 = 16C$.

The union of the plans appearing on all the contours constitutes the "plan bouquet" for the query – accordingly, plans P_1 through P_{14} form the bouquet in Figure 2. Given this set, the SpillBound algorithm operates as follows: Starting with the cheapest contour $\mathcal{IC}_1$, a carefully chosen *subset* of plans on each contour are sequentially executed *with a time limit equal to the contour's cost*. Each plan execution focuses on incrementally learning the selectivity of a specific error-prone predicate, based on the amount of data processed by the plan within its allocated time budget. This process of contour-wise plan execution ends when all the selectivities in the ESS have been fully discovered. Armed with this complete knowledge, the genuine optimal plan is now identified and used to finally execute the query to completion.

A special feature of SB, as compared to PlanBouquet, is that its contour plans are executed in "spill-mode" during the discovery process. In this mode, operator pipelines within the execution plan are selectively terminated at a chosen location, thereby ensuring that the assigned budget is maximally

[2]Filter and projection predicates on base relations are often estimated accurately with ancillary index or histogram structures.

utilized towards selectivity discovery along a particular dimension.

To make the SB methodology concrete, consider the case where the query happens to be actually located at q^*, in the intermediate region between contours $\mathcal{IC}_3$ and $\mathcal{IC}_4$, as shown in Figure 2. Assume that the optimal plan for this location, P_{q^*}, would cost $7C$ to process the query. In contrast, SB, which is unaware of the true location, would invoke the following budgeted execution sequence:

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

where the initial executions help to determine the location of q^*, and the final P_{q^*} is the ideal plan used to execute the query to completion. (For ease of visualization, the chosen subset of plans in each contour are annotated using the $\tilde{}$ symbol in Figure 2).

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

The crucial and perhaps surprising outcome of the SB strategy is that the additional execution costs entailed by its "trial-and-error" selectivity discovery exercise 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 [4], to capture the worst-case sub-optimality ratio of a query processing algorithm over the entire selectivity space. Then, the MSO of SB is bounded by

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

where D is the dimensionality of the ESS, i.e. the expected number of error-prone predicates in the input query. As a case in point, the sub-optimality of 4 in the q^* example given in Figure 2, is well within the bound of 10 guaranteed by Equation 1 for $D = 2$.

We hasten to add that, while at first glance, the MSO guarantee of Equation 1 may appear to be large in absolute terms – e.g. the guarantee for a query with a 4D ESS is 28 – it should be viewed with the perspective that the empirical MSO values obtained by contemporary database engines are typically in the several *hundreds to thousands*, as highlighted in [4, 11].

Limitation of SpillBound Notwithstanding SpillBound's unique benefits with regard to robust query processing, a major limitation is that its MSO guarantees are predicated on expending enormous pre-processing overheads during query compilation. Specifically, identifying the isocost contours in the ESS, entails in principle, $\Theta(r^D)$ calls to the query optimizer, where r is the resolution (i.e. discretization granularity) along each dimension of the ESS. So, for instance, if $r = 100$, corresponding to selectivity characterization at 1% intervals, and D is 4, a *hundred million* optimizer invocations have to be carried out to identify the contours before SB can begin executing the query. As a consequence, SB is currently suitable only for *canned queries* that are repeatedly invoked by the parent applications.

An obvious first step towards addressing the above issue is to utilize multi-core computing platforms to leverage the intrinsic parallelism available in contour identification. However, this may not be sufficient to fully address the strong exponential dependence on dimensionality. In our view, adapting the SB methodology for ad-hoc queries requires, in addition to hardware support, *algorithmic approaches* for substantive reduction of the compilation overheads – the design of such approaches forms the focus of this paper.

Problem Formulation Specifically, we investigate the trade-off between the two key attributes of the SB approach, namely, the *compilation overheads* and the *MSO guarantee*. The overhead of SB is

measured as the number of optimization calls made to the query optimizer in order to construct all the isocost contours. Given an algorithmic approach aimed at reducing this compilation overheads, we use γ (≥ 1) to denote its overheads reduction factor relative to the compilation overheads of SB. However, bringing down the compilation overheads may result in a weaker MSO guarantee. We use η (≥ 1) to denote this relaxation factor in the guarantee, relative to the MSO of SB. With this characterization, the formal problem addressed in this paper is the following:

Given a user query Q for which SpillBound *provides an MSO guarantee M, and a user-permitted relaxation factor η on this guarantee, design a query processing algorithm that maximizes γ while ensuring that the MSO guarantee remains within ηM.*

Algorithmic Reduction of the Overheads The MSO guarantees of SB are only predicated on the standard assumption of *monotonic* behavior of plan cost functions with regard to ESS predicate selectivities. In this paper, we leverage the stronger fact that plan cost functions typically exhibit a *concave down* behavior in the ESS – i.e. they have monotonically non-increasing slopes.[3] Given this behavior, we design a modified algorithm, FrugalSpillBound (FSB), which provably achieves substantive reductions in the compilation overheads at the cost of mild relaxation on the MSO guarantee. The good news is that the tradeoff between η (the relaxation factor on the MSO guarantees) and γ (the relative reduction in compilation overheads) is extremely attractive – specifically,[4]

$$\gamma = \Omega(r^D/(D \log_\eta r)^{D-1}) \qquad D \geq 2$$

In other words, we deliver an initial regime that essentially provides an *exponential improvement* in γ for a linear increase in η.

More concretely, a sample instance of the $\eta - \gamma$ tradeoff is shown in the purple line of Figure 3, obtained for a 4D ESS derived from Query 26 of the TPC-DS benchmark. In this figure, which graphs a *semi-log* plot, the initial exponential overhead reduction regime is long enough that a **two orders of magnitude improvement** in γ is achieved with an η of 2. Further, when *empirically* evaluated, the decrease in overheads is much greater – this is shown in the blue line of Figure 3, where **nearly four orders of magnitude improvement** in γ is achieved for $\eta = 2$.

The concavity assumption directly leads to an elegant FSB construction for the base case of one-dimensional ESS. However, to handle the multi-dimensional scenario, we need to bring in additional machinery, called *bounded contour-covering sets* (BCS), which serve as low-overhead replacements for the original isocost contours. A BCS is a set of locations that collectively *spatially dominate* all locations on the associated contour, and whose costs are within a bounded factor of the contour cost. Efficient identification of the BCS is made possible thanks to the concavity assumption, and we generate BCS whose aggregate cardinality over the contours is *exponentially smaller* than the number of locations in the ESS, resulting in the substantially decreased overheads.

Performance Results To demonstrate that the example $\eta - \gamma$ tradeoff for FSB shown in Figure 3 is not an isolated instance, we have carried out similar evaluations on a representative set of OLAP queries sourced from the TPC-DS benchmark, operating on the PostgreSQL engine. The query suite covers a variety of ESS dimensionalities, going up to as many as 5 dimensions, and capturing environments that are challenging from a robustness perspective. Our performance results indicate that a two orders of magnitude theoretical reduction in overheads is *routine* with $\eta = 2$, while the empirical reduction in

[3] As explained in Section 2, a weaker form of concavity, called Axis-Parallel Concavity, is sufficient for our techniques to hold.

[4] For the special case of $D = 1$, the reduction is given by $\gamma = r/\log_\eta r$, as explained in Section 3.

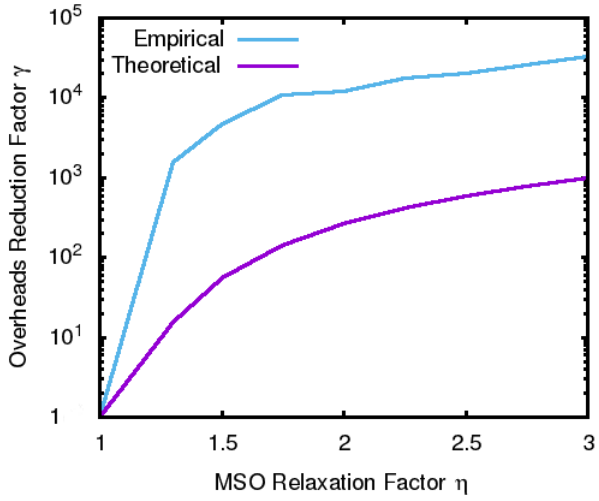


Figure 3: FSB Tradeoff for 4D_Q26

overheads is typically an order of magnitude more than this guaranteed value, delivering a cumulative benefit of more than three orders of magnitude. Therefore, in an overall sense, the new FSB approach represents a substantive step towards practically achieving robust query processing for ad-hoc queries with moderate ESS dimensionalities – this is especially so in conjunction with contemporary multi-core architectures that exploit the inherent parallelism available in the ESS construction. So, for instance, a 5D query which takes a **few days** even on a well-provisioned multi-core machine to complete the 10 billion optimizer calls required for constructing the entire ESS (at a resolution of 100), can now be made ready for execution within a **few minutes** by FrugalSpillBound!

Organization The remainder of this paper is organized as follows: In Section 2, the background to our robust query processing problem is provided, along with the assumptions and associated notations. Then, the 1D version of FSB and the associated analysis are presented in Section 3. The reworking of the FSB design for 2D ESS, via the new notion of bounded contour-covering sets, is carried out in Section 4, and the extension to arbitrary dimensions is outlined in Section 5. The experimental framework and performance results are enumerated in Section 6. Pragmatic deployment aspects and the related literature are reviewed in Sections 7 and 8, respectively. Finally, our conclusions are summarized in Section 9.

2 Background

In this background section, we review the key concepts, notations and assumptions underlying our approach to robust query processing [4, 10].

2.1 Error-prone Selectivity Space (ESS)

Given an SQL query, any predicate whose selectivity is difficult to estimate accurately is referred to as an *error-prone predicate*, or epp. For a query with D epps, the set of all epps is denoted by EPP $= \{e_1, \ldots, e_D\}$, where e_j denotes the j^{th} epp. The selectivities of the D epps are mapped to a D-dimensional space, with the selectivity of e_j corresponding to the j^{th} dimension. Since the selectivity of each predicate ranges over $(0, 1]$, a D-dimensional hypercube $(0, 1]^D$ results, henceforth

referred to as the *error-prone selectivity space*, or ESS. Note that each location $q \in (0,1]^D$ in the ESS represents a specific query instance where the epps happen to have the selectivities corresponding to q. Accordingly, the selectivity value of q on the jth dimension is denoted by $q.j$.

For tractability, the ESS 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 ESS, and the location at which the selectivity value in each dimension is maximum as the *terminus*. In our framework, the origin and the terminus correspond to query locations with $q.j = 1/r \ \forall j$ and $q.j = 1 \ \forall j$, respectively.

2.2 POSP Plans

The optimal plan for a generic selectivity location $q \in$ ESS is denoted by P_q, and the set of such optimal plans over the complete ESS constitutes the *Parametric Optimal Set of Plans* (POSP) [8].[5] We denote the cost of executing any plan P at a selectivity location $q \in$ ESS by $Cost(P, q)$. Thus, $Cost(P_q, q)$ represents the optimal execution cost for the selectivity instance located at q. Further, we assume that the query optimizer can identify the optimal query execution plan if the selectivities of all the epps are correctly known.[6] For ease of presentation, we will hereafter use *cost of a location* to refer to the cost of the optimal plan at that location.

2.3 Optimal Cost Surface (OCS)

The trajectory of the minimum cost plan through the entire D-dimensional ESS represents the Optimal Cost Surface (OCS) – an example for a 2D ESS is shown in Figure 4, where the X and Y axes represent the two join predicate selectivities, and the Z axis represents the cost of each location in this selectivity space. The intersection of the isocost hyperplanes ($\mathcal{IC}_1$ through $\mathcal{IC}_5$) with the OCS, which results in the isocost contours, is also captured in Figure 4. In fact, the projected isocost contours shown in Figure 2 were constructed from a similar OCS.

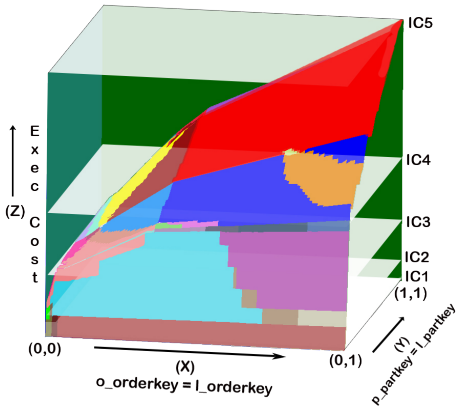


Figure 4: Optimal Cost Surface (OCS)

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

[6] For example, through the classical DP-based search of the plan enumeration space [16].

2.4 Maximum Sub-Optimality (MSO)

We now move on to the performance metrics proposed in [4] to quantify the robustness of query processing. For this purpose, let q_a denote the actual selectivities of the user query epps – note that this location is unknown at compile-time, and needs to be explicitly discovered. As discussed in the Introduction, SpillBound carries out a *sequence* of budgeted plan executions, while discovering the actual selectivities of q_a. We denote this sequence by $\texttt{Seq}_{q_a}$, with each element s_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, relative to an oracle that magically knows the correct location, and therefore uses the ideal plan P_{q_a}, is defined as:

$$SubOpt(\texttt{Seq}_{q_a}) = \frac{\sum_{s_i \in \texttt{Seq}_{q_a}} \omega_i}{Cost(P_{q_a}, q_a)} \tag{2}$$

leading to

$$MSO = \max_{q_a \in \texttt{ESS}} SubOpt(\texttt{Seq}_{q_a}) \tag{3}$$

That is, the MSO represents the worst-case suboptimality that can occur with regard to plan performance over the entire ESS space.

2.5 Plan Cost Monotonicity (PCM)

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 $q_1, q_2 \in$ ESS, 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 ESS is that the cost functions of the plans appearing in the ESS 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 $q_b, q_c \in$ ESS, and for any plan P,

$$q_b \succ q_c \Rightarrow Cost(P, q_b) > Cost(P, q_c) \tag{4}$$

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*.

We augment the above assumption with a stricter condition in this paper, wherein not only are the PCFs monotonic, but also exhibit a weak form of *concavity* in their cost trajectories, as explained next.

2.6 Axis-Parallel Concavity (APC)

In a one-dimensional world, a plan cost function $\mathcal{F}_p$ is said to be concave if, for any pair of locations q_1, q_2 in the 1D ESS, and any $\alpha \in [0, 1]$,

$$\mathcal{F}_p((1-\alpha)q_1 + \alpha q_2) \geq (1-\alpha)\mathcal{F}_p(q_1) + \alpha \mathcal{F}_p(q_2)$$

Extending to the general case of D dimensions, a PCF $\mathcal{F}_p$ is said to be *axis-parallel concave* (APC) if the function is concave along every axis-parallel 1D segment of the ESS. That is, the above equation is satisfied by any pair of locations q_1, q_2 in the ESS that belong to a common 1D segment of the ESS (i.e., $\exists j$ s.t. $q_1.k = q_2.k, \forall k \neq j$).

For example, if e_1 and e_2 are the epps of a 2D ESS, then the APC requirement is that each PCF should be concave along every vertical and horizontal line in the ESS.

Note that APC is a strictly *weaker* condition than complete concavity across all dimensions – that is, all fully-concave functions automatically result in APC, but the reverse may not be true. An important implication of APC of the PCFs is that it is easily provable that the corresponding OCS, which is the infimum of the PCFs, *also satisfies APC*. Finally, for ease of presentation, in the remainder of this paper, we will generically use concavity to mean APC.

Empirical Validation of APC

An immediate question that arises in the above context is whether the concavity assumptions on the PCFs (and, by implication, the OCS) generally hold true in practice. For this purpose, we have carried out extensive experimental evaluation with the TPC decision-support benchmarks operating on contemporary database engines. The summary finding of this empirical evaluation, whose details are presented later in Section 6, is that APC is consistently observed over almost the entire ESS.

As a sample instance, the axis-parallel projections of the 2D OCS presented in Figure 4, are computed in Figures 5a and 5b for the $Orders \bowtie Lineitem$ and $Part \bowtie Lineitem$ join predicates, respectively. These figures are graphed on a *log-log scale* and for ease of representation, capture only the optimality region of each PCF. We observe here that the PCFs clearly exhibit concavity in their optimality regions with respect to selectivity. As a direct consequence, the OCS exhibits concavity over the entire selectivity range, justifying the assumption on which our results are based. Moreover, given current relational operator cost functions, a detailed rationale as to why PCF and OCS concavity is to be expected, is discussed below.

Rationale of APC Let us now see the rationale behind why we can expect axis-parallel concave behaviour of PCFs. Since a plan is a tree of operators, the individual operator cost behaviours are as follows: Except for the sort operator, we observed that all the operators are having non-increasing slope, piecewise linear or simply linear behaviour.

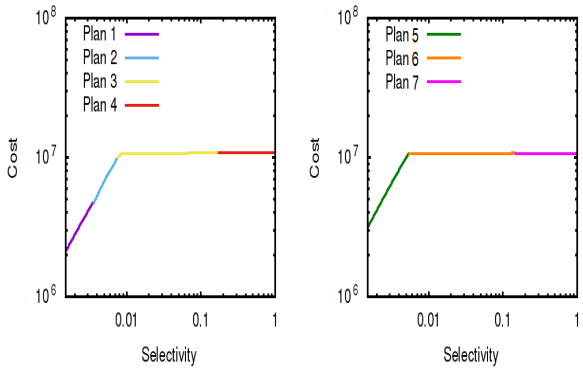
Starting with join operators, Hash Join and Merge Join (in case of already sorted inputs) costs $\mathcal{O}(s_x + s_y)$ where s_x, s_y represents the two input selectivities from its upstream operators. As we are interested in 1D axis-parallel projections, wherein one of s_x or s_y is constant, we can expect the two operators to behave linearly. Similar is the case with Nested Loop Join in which case it costs $\mathcal{O}(s_x * s_y)$. Shifting the focus to scanning operators, Sequential Scan has constant cost, whereas Index Scan and Bitmap Scan is linear. Finally, Sort operator has cost of the form $\mathcal{O}(s_x \log(s_x))$ which is superlinear. We can expect axis-parallel concave behaviours of PCFs, based on the following observations:

1. From [10], for a given plan, we know that the total cost of a plan is sum of cost of its constituent operators

2. In practice, contribution from Sort is very less compared to the total cost of a plan in industrial strength benchmarks [5]

3. Point-wise sum of a set of concave functions is also a concave function

OCS concavity follows from the fact that OCS is a infimum of all POSP PCFs, and that infimum of axis-parallel concave functions are also axis-parallel concave. Note that for the working of the FrugalSpillBound algorithm, we just need OCS to be axis-parallel concave.



(a) Projection on X dimension (b) Projection on Y dimension

Figure 5: Validation of Axis-Parallel Concavity

2.7 Compilation Overheads

As mentioned in the Introduction, we measure the query compilation overheads in terms of the number of optimization calls made to the underlying database engine. With regard to this metric, the overheads incurred by SB in constructing the ESS can be computed as follows: SB first computes the optimal plans for *all* locations in the discretized ESS grid. This is carried out through repeated invocations of the optimizer with different selectivity values and combinations. Then, the isocost contours are drawn as connected curves on this discretized diagram. So, if we assume a grid resolution of r in each dimension of the ESS, the total number of optimization calls required by this approach is r^D.

Note, however, that we do not require the complete characterization of the ESS, but only the parts related to the isocost contours, as shown in Figure 2. An optimized variant, called **Nexus**, was proposed in [4] to implement this observation, and shown to make material reductions in the contour identification overheads. However, we have not included Nexus in our current study for the following reasons: (1) When a large number of contours are present in the ESS, which can happen if the cost at the ESS terminus is much larger than at the origin, the net effort by Nexus in contour identification effectively becomes close to complete enumeration. (2) If a lower bound on a query's location in the ESS happens to be known through domain knowledge, SpillBound can take advantage by making the lower bound to be the origin and thereby shrinking the ESS. However, the isocost contours would have to be redrawn from scratch by Nexus. (3) To provide performance fairness to queries across the ESS, *randomized* placement of contours was presented as a solution in [4]. In such cases, multiple sets of contours would have to be identified by Nexus, and it may cumulatively turn out to be more expensive as compared to complete enumeration. We have therefore chosen to instead simply assume that the entire ESS is enumerated, and consequently r^D is used as the baseline SB overheads in the sequel. Further, note that FrugalSpillBound is *not* impacted by such deployment issues since its compile-time efforts are carried out afresh at each ad-hoc query's submission time.

Notation Summary: For easy reference, the notations used in the remainder of the paper are summarized in Table 1.

3 Frugal SpillBound for 1D ESS

In this section, we consider the baseline case of a 1D ESS. The sample concave OCS function $\mathcal{F}$, shown in Figure 6, is used to aid the description. In this figure, the X-axis represents the selectivity

Table 1: NOTATIONS

Notation	Meaning
epp (EPP)	Error-prone predicate (its collection)
ESS	Error-prone selectivity space
D	Number of dimensions in the ESS
r	Grid resolution in each ESS dimension
$e_1, \ldots, e_D$	The D epps in the query
$q \in [0,1]^D$	A query location in the ESS
$q.j$	Selectivity of q in jth ESS dimension
P_q	Optimal Plan at q
q_a	Actual query location in ESS
$Cost(P,q)$	Cost of plan P at location q
$\mathcal{IC}_i$	Isocost Contour i
$\texttt{CC}_\texttt{i}$	Cost of an isocost contour $\mathcal{IC}_i$
BCS_i	Bounded contour-covering set of contour $\mathcal{IC}_i$
η	User-specified MSO relaxation factor
γ	Reduction factor wrt compilation overheads

range for the lone epp and the Y-axis represents the OCS function. The Y-axis is discretized into doubling-based isocost contours, $\mathcal{IC}_1$ through $\mathcal{IC}_m$, with $\texttt{CC}_\texttt{i} = 2^{i-1}C$. Note that, in the case of 1D OCS, each of the contours correspond to a *single* selectivity location on the X-axis. We denote the location corresponding to $\mathcal{IC}_i$ by Q_i. Further, $Q_1 = 1/r$ and $Q_m = 1$ correspond to the origin and terminus locations, respectively.

1D SB We begin by reviewing how the SpillBound algorithm works on the above 1D ESS. Conceptually, the algorithm has two phases, a *compilation phase* and an *execution phase*. During the compilation phase, for each of the r uniformly spaced locations on the X-axis, the optimal plan at the location and its cost are determined. Using the cost information from the r locations, the precise location of Q_i is identified for $i = 1, \ldots, m$. The set of optimal plans at the Q_i locations is called the "bouquet of plans". During the execution phase, this bouquet of plans is sequentially executed, starting from the cheapest isocost contour, with a budget equal to the associated contour cost. The process ends when a plan reaches completion within its allocated budget. As proved in [4], this budgeted sequence of plan executions guarantees achieves an MSO of 4 with a compilation overhead of r optimizer calls.

We now move on to presenting the 1D FSB algorithm, which also has compilation and execution phases, as described below.

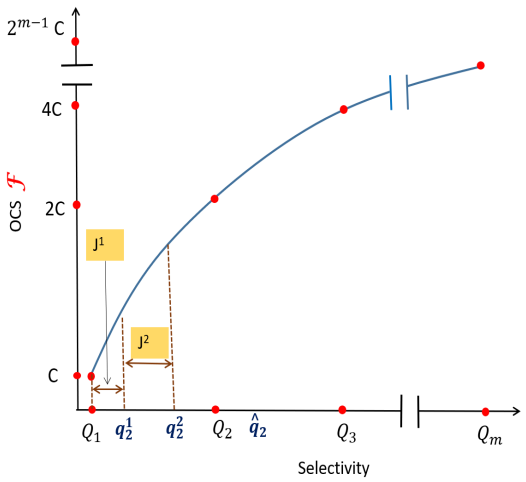


Figure 6: Concave OCS

3.1 Compilation Phase

The main idea in the compilation phase of FSB is to do away with SB's approach of precisely identifying the location of the Q_is. Instead, for each Q_i, we identify a *proxy location*, $\hat{q}_i$, such that the cost of the optimal plan at $\hat{q}_i$ is in the range $[\mathcal{F}(Q_i), \eta\mathcal{F}(Q_i)]$. The compilation phase consists of identifying these proxy locations $\hat{q}_i$s via a sequence of calibrated *jumps* in the selectivity space, as described next.

Discovering the proxy for Q_2

Since Q_1 is known, we set $\hat{q}_1 = Q_1$. The search for $\hat{q}_2$ starts from $\hat{q}_1$. We now perform a sequence of jumps in the selectivity space until we land exactly at Q_2 or overshoot it for the first time. Further, each jump is calibrated such that the cost of $\hat{q}_2$ is guaranteed to be in the range $[\mathcal{F}(Q_2), \eta\mathcal{F}(Q_2)]$, as described below.

First Jump Identify the optimal plan $P_{\hat{q}_1}$ at $\hat{q}_1$, and compute its slope, $s(\hat{q}_1)$ at $\hat{q}_1$.[7] The slope is calculated through plan recosting[8] of $P_{\hat{q}_1}$ at a selectivity location in the close neighborhood of $\hat{q}_1$.

Our first estimate for $\hat{q}_2$, denoted by q_2^1 (refer to Figure 6), is the location that is expected to have η times the cost of $\mathcal{F}(\hat{q}_1)$ when extrapolated by a tangent line with a slope of $s(\hat{q}_1)$, i.e,

$$\frac{\mathcal{F}(\hat{q}_1) + s(\hat{q}_1) \cdot (q_2^1 - \hat{q}_1)}{\mathcal{F}(\hat{q}_1)} = \eta \tag{5}$$

By rearranging, we get

$$\begin{aligned} q_2^1 &= \hat{q}_1 + \frac{(\eta - 1) \cdot \mathcal{F}(\hat{q}_1)}{s(\hat{q}_1)} \\ q_2^1 &= \hat{q}_1 + J^1 \end{aligned} \tag{6}$$

where J^1 represents the first jump towards $\hat{q}_2$, relative to the starting location, $\hat{q}_1$. The following lemma immediately follows from the concavity of the PCF.

[7] The slope at any location in $\mathcal{F}$ is > 0 due to PCM.

[8] This feature enables costing of an abstract plan for a query, and is around a hundred times faster than optimizer calls [5].

Lemma 3.1 *The cost condition $\mathcal{F}(q_2^1) \leq \eta \cdot \mathcal{F}(\hat{q}_1)$ is satisfied at q_2^1.*

We next show that the jump J^1 is such that the selectivity of q_2^1 is at least η times the selectivity at $\hat{q}_1$.

Lemma 3.2 *The selectivity of q_2^1 is at least η times the selectivity at $\hat{q}_1$, i.e, $\boldsymbol{q_2^1 \geq \eta \hat{q}_1}$*

Proof 1 *Let the tangent of $\mathcal{F}$ at $\hat{q}_1$ be expressed as*

$$\mathcal{F}(q) = s(\hat{q}_1) \cdot q + c' \qquad 0 \leq q \leq 1, c' \geq 0$$

Here, $c' \geq 0$ to ensure non-negative cost at $q = 0$. We get

$$\mathcal{F}(\hat{q}_1) = s(\hat{q}_1) \cdot \hat{q}_1 + c'$$
$$\eta \cdot \mathcal{F}(\hat{q}_1) = s(\hat{q}_1) \cdot q_2^1 + c'$$

Rearranging the above two equations, we get

$$\eta s(\hat{q}_1) \cdot \hat{q}_1 + \eta c' = s(\hat{q}_1) \cdot q_2^1 + c'$$
$$q_2^1 = \eta \hat{q}_1 + \frac{(\eta - 1)c'}{s(\hat{q}_1)}$$
$$\boldsymbol{q_2^1 \geq \eta \hat{q}_1}$$

Depending upon the cost at the first jump's landing location, i.e. at q_2^1, there can be two cases:

1. *Cost Overshoot*, i.e, $\mathcal{F}(q_2^1) \geq \mathcal{F}(Q_2)$: In this case, we have identified a proxy location for Q_2 whose cost is at most $\eta\mathcal{F}(Q_2)$ (by Lemma 3.1).
2. *Cost Undershoot*, i.e., $\mathcal{F}(q_2^1) < \mathcal{F}(Q_2)$: In this case, the above-mentioned jump scheme is repeated with q_2^1 as the starting location. That is, we jump to q_2^2, with the jump length being $J^2 = \frac{(\eta - 1)\mathcal{F}(q_2^1)}{s(q_2^1)}$. This process is repeated until we reach $\hat{q}_2$, signalled by $\mathcal{F}(\hat{q}_2) \geq \mathcal{F}(Q_2)$. Since the cost of $\hat{q}_2$'s previous location is less than $\mathcal{F}(Q_2)$, Lemma 3.1 guarantees that $\mathcal{F}(\hat{q}_2) \leq \eta\mathcal{F}(Q_2)$.

3.1.1 Implementation of Proxy Discovery

The compilation phase of FSB for the 1D scenario is detailed in Algorithm 1. Here, the entire search from $\hat{q}_1$ to $\hat{q}_2$ is captured as a generic Explore subroutine, with three arguments: seed, the starting location, t_cost, the cost at the terminal location, and r_factor, the relaxation factor wrt t_cost.

The proxy location $\hat{q}_i$ for Q_i is obtained starting with the proxy location $\hat{q}_{i-1}$. This is done by calling the Explore subroutine, with seed as $\hat{q}_{i-1}$, target cost of $\texttt{cc}_\texttt{i}$, and relaxation factor of η. The argument that bounded the relative cost of $\hat{q}_2$ w.r.t. the cost of Q_2 can be repeated to show that the cost of $\hat{q}_i$ is at most $\eta\mathcal{F}(Q_i)$ for $i = 2, \ldots, m-1$. Finally, the output of the algorithm is a set of proxy locations, ProxyContourLocs = $\{Q_1 \cup \{\bigcup_{i=2}^{m-1} \hat{q}_i\} \cup Q_m\}$.

3.1.2 Bounded Compilation Overheads

Theorem 3.3 *The compilation overheads reduction, γ, of 1D* FSB *is at least $\frac{r}{\log_\eta r}$.*

Proof 2 *From Lemma 3.2, the maximum number of jumps is required when the selectivity estimation at each jump is exactly η times the selectivity of the previous location. Therefore, the total number of query optimizer calls is bounded as follows.*

$$\textit{Total Optimization calls } \leq \log_\eta \frac{Q_m}{Q_1} \leq \log_\eta r$$

Thus, the compilation overheads come down from r to $\log_\eta r$.

Algorithm 1 1D `FrugalSpillBound` (η)

Compilation Phase:
set $Q_1 = 1/r$ and $Q_m = 1$;
set $k = 2$;
set ProxyContourLocs = $\{Q_1, Q_m\}$;
set $\hat{q}_1 = Q_1$;
while $k < m - 1$ **do**
    $\hat{q}_k$ = Explore($\hat{q}_{k-1}$, $\texttt{CC}_\texttt{k}$, η);
    Add $\hat{q}_k$ to ProxyContourLocs;
    k++;
end while

function Explore(seed, t_cost, r_factor);
compute $cost = \mathcal{F}$(seed) (using optimizer call);
while $cost <$ t_cost **do**
    compute $slope$ at seed (using plan recosting);
    $next\_jump = (\text{r\_factor} - 1) \cdot \dfrac{cost}{slope}$;
    seed $+= next\_jump$;
    $cost = \mathcal{F}$(seed);
end while
return seed;
end function

Execution Phase:
for q in ProxyContourLocs **do**
    Execute optimal plan P_q with budget $\mathcal{F}(q)$;
    if P_q completes execution **then**
        Return query result;
    else
        Terminate P_q and discard partial results;
    end if
end for

3.2 Execution Phase

The execution phase of `FSB`, as shown in Algorithm 1, is the same as that of `SB` with the plan bouquet now consisting of the optimal plans at the proxy locations in ProxyContourLocs. Therefore, the analysis needs to simply take into account that the cost of a proxy location $\hat{q}_i$ is at most η times the cost of Q_i. Thus we have the following theorem for *maintaining the η constraint*.

Theorem 3.4 *The MSO relaxation of 1D* `FSB` *is at most η.*

Proof 3 *The bounded cost of each proxy location ensures that the sequence of execution costs for the "bouquet of plans" for 1D* `FSB` *is $C, 2\eta C, 4\eta C, \ldots$ (as opposed to $C, 2C, 4C, \ldots$ for* `SB`*). Since the MSO of* `SB` *is 4, it follows that the MSO of 1D* `FSB` *is bounded by 4η.*

4 Frugal SpillBound for 2D ESS

In this section, we present the extension of 1D FSB to the 2D case. For ease of exposition, we refer to the two epps as x and y, respectively.

In the 1D ESS, each contour was a single point. However, in 2D, it is a continuous 1D curve as shown in Figure 7. Therefore, the step of identifying the proxy locations for Q_is has to be generalized so as to cover an isocost contour $\mathcal{IC}_i$ with an appropriate set of proxy locations. We achieve this by finding a *bounded contour-covering set* (BCS) of locations for each contour $\mathcal{IC}_i$. The definition of these sets and the process for identifying them is presented next.

4.1 Bounded Contour-covering Set (BCS)

The BCS for a contour is defined as the *minimal set* of locations such that every location in the contour is *spatially dominated* by at least one location in this set. Further, the cost of each location in BCS is required to be within an η factor of the contour cost. We denote the BCS of contour $\mathcal{IC}_i$ by BCS_i. Formally, BCS_i is a minimal set that needs to satisfy the following condition:

$\forall q \in \mathcal{IC}_i, \exists\, q' \in BCS_i$ such that

$$q \preceq q' \text{and } Cost(P_{q'}, q') \leq \eta \texttt{CC}_\texttt{i} \quad (7)$$

For the example contour $\mathcal{IC}_i$ shown in Figure 7, a candidate BCS_i is $\{c_1, c_2, c_3\}$ which covers the entire contiguous region of the contour. As a case in point, the covering location c_2 fully covers the optimality segments of P_6 and P_5, as well as parts of P_7 and P_4, in $\mathcal{IC}_i$.

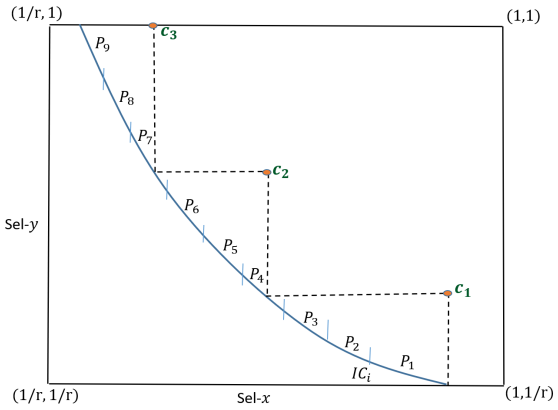


Figure 7: Covering Contour

4.2 Compilation Phase

We now present a computationally efficient method to find a BCS for an isocost contour in the 2D ESS. To generalize the 1D method, we carry out jumps in the selectivity space along *both* the x and y dimensions. These jumps are designed to be axis-parallel and we leverage APC in their analysis. Further, the jumps in the two dimensions are in *opposite* directions – *forward* in one, and *reverse* in the other. The term "reverse" refers to the fact that the jumps are performed in the decreasing selectivity direction.

Assume that the search steps in the x dimension are reverse jumps. This means that for each jump, the selectivity of the next location in the x dimension is decreased by a constant factor. On the other hand, in the y dimension, the jumps are forward, and at each step, the Explore (seed,t_cost,r_factor) subroutine is invoked to decide the next location.

4.2.1 Algorithm Description

We present the algorithm by describing the process of constructing BCS_i for the isocost contour $\mathcal{IC}_i$ shown in Figure 7. For ease of presentation, we refer to Figure 8 which overlays the construction of BCS_i on top of contour $\mathcal{IC}_i$. The main idea is to carry out a sequence of interleaved search steps that alternatively explore the x and y dimensions.

The algorithm starts from the location $c_0 = (1, 1/r)$ as the seed. We search for a location, u_1, on $y = 1/r$ line whose cost is in the range $[\texttt{CC}_\texttt{i}, \sqrt{\eta}\texttt{CC}_\texttt{i}]$. A sequence of reverse jumps from c_0 with $\sqrt{\eta}$ factor decrease in selectivity is carried out until we reach u_1. The Explore subroutine is now invoked along the increasing y dimension with u_1 as the seed location, terminating cost $\sqrt{\eta}\texttt{CC}_\texttt{i}$, and relaxation factor $\sqrt{\eta}$. Let the location returned be c_1, and by the construction of Explore, we know that its cost is in the range $[\sqrt{\eta}\texttt{CC}_\texttt{i}, \eta\texttt{CC}_\texttt{i}]$. Now, starting from c_1, a sequence of reverse jumps, again with $\sqrt{\eta}$ decrease in selectivity, are carried out till we reach a location u_2 whose cost is in the range $[\texttt{CC}_\texttt{i}, \sqrt{\eta}\texttt{CC}_\texttt{i}]$. This is followed by a call to Explore with u_2 as the seed and the same settings as before for the other arguments. The returned location is now c_2. This interleaved process of reverse jumps along the x dimension and forward jumps along the y dimension, is repeated until the process hits the boundary of the ESS. Let us say that the process ends at location c_k. Then, the set $BCS_i = \{c_1, \ldots, c_k\}$ is returned as the BCS of contour $\mathcal{IC}_i$ ($k = 4$ for the example contour in Figure 8). This description of the compilation phase of 2D FSB is codified in Algorithm 2.

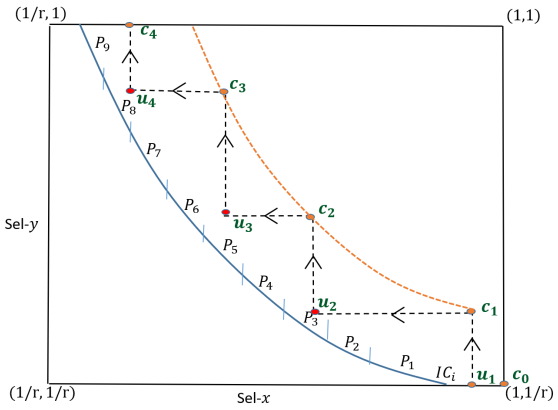


Figure 8: Covering Set Identification

4.2.2 Proof of Correctness

In order to show that every location in the contour is spatially dominated by a location in the bounded contour-covering set, we need to first prove that reverse jumps allow us to find u_is, whose cost is in the range $[\texttt{CC}_\texttt{i}, \sqrt{\eta}\texttt{CC}_\texttt{i}]$. Note that this is true if each reverse jump results in a relative cost decrease of at most $\sqrt{\eta}$. To prove this, fix a covering location c_k, and let $\mathcal{F}_{ap}$ denote the restriction of OCS to the horizontal line passing through c_k. Then,

Lemma 4.1 *The reverse jump from a location q along the x direction by a factor $\sqrt{\eta}$ results in a relative cost decrease of at most $\sqrt{\eta}$, i.e., $\mathcal{F}_{ap}(\frac{q.x}{\sqrt{\eta}}, q.y) \geq \mathcal{F}_{ap}(q.x, q.y)/\sqrt{\eta}$.*

Proof 4 *We know from APC that $\mathcal{F}_{ap}$ is concave. Let q' denote the location $(q.x/\sqrt{\eta}, q.y)$. Consider the line passing through q', parallel to the X-axis, and tangent to OCS. Let c' be its Y-intercept and s be its slope. Note that $c' \geq 0$ to ensure non-negative cost near the origin. As $\mathcal{F}_{ap}$ is concave and increasing, this line* overestimates *the cost at q. Thus,*

$$\begin{aligned}
\mathcal{F}_{ap}(q) &\leq s \cdot (q.x) + c' \\
&\leq \sqrt{\eta}\left(s \cdot (\frac{q.x}{\sqrt{\eta}}) + \frac{c'}{\sqrt{\eta}}\right) \\
&\leq \sqrt{\eta}\left(s \cdot (\frac{q.x}{\sqrt{\eta}}) + c'\right) \\
&= \sqrt{\eta}\mathcal{F}_{ap}(q') \qquad (8)
\end{aligned}$$

where the second and third inequalities are implied by $\eta \geq 1$ and $c' \geq 0$. The last equality follows from the fact that the line passes through q'.

Lemma 4.2 *Every location in $\mathcal{IC}_i$ is dominated by at least one location in BCS_i .*

Proof 5 *Consider any point q in $\mathcal{IC}_i$. By construction we know that there exist $c_k \in BCS_i$ s.t. $c_k.y \leq q.y \leq c_{k+1}.y$. We will show that $c_{k+1} \in BCS_i$ is a dominating location for q by proving $q.x \leq c_{k+1}.x$. Consider the location u_{k+1} whose x coordinate is the same as that of c_{k+1}. This means that (a) $u_{k+1}.x = c_{k+1}.x$, and (b) $u_{k+1}.y = c_k.y$. Since the cost of location u_{k+1} is greater than or equal to the cost of location q, and $u_{k+1}.y \leq q.y$ by PCM, it implies that $u_{k+1}.x \geq q.x$. Therefore, q is dominated by c_{k+1}.*

4.2.3 Bounded Computational Overheads

Lemma 4.3 *The overheads reduction, γ, of 2D* FSB *is at least* $\dfrac{r^2}{4 \cdot m \cdot \log_\eta r}$.

Proof 6 *Let us first look at the number of optimization calls required per contour for 2D* FSB*. We know that the exploration of c_ks and u_ks move unidirectionally along the y-axis ($1/r$ to 1) and x-axis (1 to $1/r$), respectively. Furthermore, we have earlier shown that each jump results in a relative increase (or decrease) in selectivity of at least $\sqrt{\eta}$. Thus, by geometric progression, we can infer the following:*

$$\begin{aligned}
\textit{Opt. calls per Contour} &= \textit{Opt. calls for } c_k\textit{s} + \textit{Opt. calls for } u_k\textit{s} \\
&\leq \log_{\sqrt{\eta}} r + \log_{\sqrt{\eta}} r \\
&= 2\log_{\sqrt{\eta}} r \\
&= 4 \cdot \log_\eta r \qquad (9)
\end{aligned}$$

Since there are m contours in the ESS*, we conclude that there are $4 \cdot m \cdot \log_\eta r$ optimization calls across all contours for 2D* FSB*, as compared to r^2 for 2D* SB*. Thus, γ is at least* $\dfrac{r^2}{4 \cdot m \cdot \log_\eta r}$.

4.3 Execution Phase

In the execution-phase, we run the original 2D SB algorithm treating the BCS identified for every contour as the effective contour. In particular, in every BCS_i, starting from the least cost BCS, plans corresponding to locations in BCS_i are executed as per the 2D SB algorithm for a contour. This contour-wise execution of plans is continued until the actual selectivities of both the epps are learned. Finally, the optimal plan is executed to compute the query results for the user.

Algorithm 2 2D `FrugalSpillBound` Algorithm (η)

Compilation Phase:
Set: $q_{cur} = (1/r, 1/r)$;
Set: $\beta = \sqrt{\eta}$;
while contours are remaining **do**
  */*Let $\mathcal{IC}_i$ denote current contour and $\texttt{CC}_\texttt{i}$ be its cost*/*
  while $q_{cur}.x \geq \frac{1}{r}$ and $q_{cur}.y \leq 1$ **do**
    Find u_i with cost in $[\texttt{CC}_\texttt{i}, \beta \cdot \texttt{CC}_\texttt{i}]$, by x-axis reverse jumps;
    $q_{cur}.x = u_i.x$;
    Call Explore$(u_i, \beta \cdot \texttt{CC}_\texttt{i}, \beta)$ along y-axis to find c_i;
    $q_{cur}.y = c_i.y$;
  end while
  Union of all c_is forms the bounded contour-covering set, BCS_i;
  / Move to next contour */*
end while

Execution Phase:
Run the original 2D `SpillBound` algorithm on the plans corresponding to BCS_i, of every contour $\mathcal{IC}_i$;

4.3.1 Maintaining the η constraint

Theorem 4.4 *The MSO relaxation of 2D* `FSB` *is at most* η.

Proof 7 *We know that the cost of any location in BCS_i is at most $\eta\texttt{CC}_\texttt{i}$. Furthermore, the execution-phase runs the 2D* `SB` *algorithm on the BCS of every contour. Thus, every execution in 2D* `FSB` *is performed with a budget of η times its corresponding contour cost. Hence, the overall cost of 2D* `FSB` *is at most η times that of 2D* `SB`*, which increases the MSO by at most η.*

The analysis of the 2D `SpillBound` algorithm in [10] relied on two crucial properties: *Half-Space Pruning* (HSP) and *Contour Density Independent Execution* (CDIE). HSP says that a single spill-mode execution of a plan is sufficient to divide the `ESS` into two disjoint half-spaces and obtain evidence that q_a does not lie in one of the two half-spaces (i.e. one of the half-spaces gets pruned). On the other hand, the CDIE property implies that the number of executions required per contour is independent of the number of plans on the contour and is bounded by D. Both the HSP and CDI Execution properties continue to hold for 2D `FSB` also, and this is formally proved below.

4.3.2 Half-Space Pruning and Contour Density Independent Execution

As mentioned before, `SB` is predicated on two building blocks namely: a) Half-Space Pruning (HSP), and b) Contour Density Independent (CDI) executions (Section 3 of [10]). With regard to identifying the set of plans to be executed and which `epp` to spill on, `SB` and `FrugalSpillBound` employ the same procedure and this observation is sufficient to establish the CDI property of `FrugalSpillBound`. We now show that HSP is also satisfied.

The half-space pruning property is achieved by leveraging the notion of *"spilling"*, whereby operator pipelines are prematurely terminated at chosen locations in the plan tree, in conjunction with run-time monitoring of operator selectivities. The following lemma ([10]) is sufficient to prove the half-space pruning property.

Lemma 4.5 *Consider a location $q \in$ ESS and a plan P_q which is optimal at q. Let e_j be the* epp *identified to spill for P_q. When P_q is executed with budget $Cost(P_q, q)$ and spilling on e_j, then we either learn (a) the exact selectivity of e_j, or (b) that $q_a.j > q.j$.*

The proof idea is that total cost of a plan tree is essentially sum cost of its individual operator nodes. e_j is selected such that its upstream operators are error-free from selectivity estimates. The proof of the lemma follows because of the following reasons: a) due to spilling, e_j's downstream operators are not executed, b) cost of e_j's upstream operators are accurately known and accounted in $Cost(P_q, q)$, and c) $Cost(P_q, q)$ also additionally accounts for $q.j$ selectivity of e_j.

In SB, at any point in time, if a location q is being explored, it is always ensured that the selectivity value of q along non-epp dimensions is set to their selectivity at q_a. However, when we replace a continuous isocost contour with a discrete covering set, as in the case of FrugalSpillBound, this may not always be sufficient. However, it is true that, when a location q is explored in FrugalSpillBound, for a non-epp dimension j, we ensure that $q.j \geq q_a.j$. This is sufficient to show that the proof of Lemma 4.5 applies to FrugalSpillBound as well.

4.3.3 Contour Covering Set identification

In the original SpillBound (SB) algorithm, once a selectivity is completely learnt for an epp, the original ESS gets projected on the selectivity value of the learnt dimension. This process of reducing the dimensioanlity of the ESS by 1 continues as and when an epp get completely learnt, until the actual selectivity of all the epps are discovered.

The isocost contours in the SB are continuous whereas the contour covering sets are discrete sets. Therefore, the update to the effective ESS by projecting the current ESS onto the selectivity of the learnt dimension needs to be done carefully in the case of FrugalSpillBound. The sensitive situation is when the learnt selectivity value is such that, there is no location in the covering set whose value on the learnt dimension is exactly equal to the learnt value. For example, in Figure 8, say that FSB learns the complete or actual selectivity in the Y-dimension first, whose value is strictly in between $u_3.y$ and $c_3.y$, i.e. $u_3.y < q_a.y < c_3.y$. In this case, we have to take care to ensure that, we project the 2D ESS onto the line $y = c_3.y$ to ensure that there is a valid starting locations for the 1D search along x dimension. This notion also extends naturally when we are doing projections in the multi-dimensional case.

5 Multi-Dimensional FSB

In this section, we show how to extend 2D FSB to higher dimensions, and present Multi-D FSB for this purpose. The number of potential epp in a canonical OLAP query is usually large – for instance, we have observed as many as 12 for some TPC-DS queries. If all these epps were made part of the ESS, it would result in an impractically large search space that cannot be explored efficiently. Therefore, before describing the Multi-D FSB algorithm, we first explain how it is usually feasible to construct a low-dimensional ESS from the initial large set, through a pre-processing step, *without* impacting the MSO guarantees of the query.

5.1 Constructing Low-Dimensional ESS

Our Dimension Reduction preprocessor, **DimRed**, aims to prune the initial exhaustively enumerated set of ESS dimensions to a much smaller relevant set of dimensions. We only summarize the approach here due to space constraints – the complete details are available in [15].

The key idea in DimRed is to associate an *impact factor* with each potential epp. Given an epp e, its impact factor is defined as the worst case relative inflation in the cost of POSP plans when the selectivity of e is varied from $1/r$ to 1, while keeping the selectivities of other predicates fixed. Now consider a predicate e with impact factor f. If we drop e from the EPP, the induced ESS can be sub-optimal by a factor $(1 + f)$ w.r.t. the ESS that contains e. Therefore, we consider the epps in the increasing order of their impact factors, and incrementally keep dropping them while the net benefit of the reduced dimensionality on the MSO guarantee is more than the net sub-optimality incurred by the dropped predicates. The final set of retained predicates constitutes the relevant ESS for the Multi-D FSB.

We hasten to add that, as shown in [15], the impact factor of an epp can be computed efficiently, and this extends to the entire reduction algorithm. For instance, the dimensionality of TPC-DS Q91 was reduced by DimRed from 12 to 5 in around 15 seconds.

5.2 Multi-D Algorithm

The Multi-D FSB algorithm is run on the set of dimensions retained after the above pre-processing step. These retained epps are ordered in decreasing value of their impact factors, i.e, e_1 has the highest impact factor and e_D, the lowest. The basic idea behind the generalization to Multi-D FSB is to recursively call 2D FSB for the last two dimensions (i.e, those with lowest impact factors), while carefully freezing the values for the remaining $D-2$ dimensions. Specifically, the value for each of these $D-2$ dimensions is initially set to $1/r$, and then iteratively increased by a factor $\beta = \sqrt[D]{\eta}$. After the values are frozen, then the 2D FSB algorithm searches for a 2D contour whose cost is β^{D-2} times the contour's cost, and covers it with an increased factor of β^2. The value for β is chosen such that the resultant MSO increases by atmost an η factor. The pseudocode for Multi-D FSB is presented in Algorithm 3.

5.3 Proof of Correctness

We shall begin with introducing the notion of a *sub-contour*, denoted by $\mathcal{IC}|_E^H$, for any contour $\mathcal{IC}$. It is defined to be the set of locations that belong to intersection of *hyperplane* H and $\mathcal{IC}$, and then projected on the dimensions $E \subseteq$ EPP. Note that any location $q \in \mathcal{IC}|_E^H$ would be a $|E|$ dimensional location. Furthermore, a sub-contour, $\mathcal{IC}_{i_1}|_E^H$, *covers* another sub-contour $\mathcal{IC}_{i_2}|_E^H$, if $\nexists$ a location $q \in \mathcal{IC}_{i_2}|_E^H$ s.t. q dominates[9] some location of $\mathcal{IC}_{i_1}|_E^H$. In words, there should not exist a location in $\mathcal{IC}_{i_2}|_E^H$ which dominates some location in $\mathcal{IC}_{i_1}|_E^H$. Furthermore, for any location $q \in$ ESS, we use $q|_E$ to denote the projection of q on dimensions E. Notation $[n], n \in \mathcal{Z}^+$ is used represent the set of integers from $\{1, \cdots, n\}$. Finally, we can also represent a location $q \in$ ESS by its direct sum of its hyperplanes, $q_1|_{[D-2]}$ and $q_2|_{D-1,D}$ for some $q_1, q_2 \in$ ESS. Denoting the direct sum by $\circ$, then

$$q := q_1|_{[D-2]} \circ q_2|_{D-1,D} \Leftrightarrow$$
$$q.j = q_1.j, j \in [D-2] \text{ and } q.j = q_2.j, j \in \{D-1, D\}$$

Lemma 5.1 *For some two locations $q_1, q_2 \in \mathcal{IC}$ and $q_2|_{[D-2]} \succ q_1|_{[D-2]}$, the 2D contour obtained by intersecting the hyperplane $q_1|_{[D-2]}$ with $\mathcal{IC}$, covers the one obtained by intersecting $\mathcal{IC}$ with $q_2|_{[D-2]}$. That is, sub-contour $\mathcal{IC}|_{D-1,D}^{q_1|_{[D-2]}}$ covers $\mathcal{IC}|_{D-1,D}^{q_2|_{[D-2]}}$.*

Proof 8 *For sake of contradiction, let us say that sub-contour $\mathcal{IC}|_{D-1,D}^{q_1|_{[D-2]}}$* does not *cover $\mathcal{IC}|_{D-1,D}^{q_2|_{[D-2]}}$. Further, this means there exist a 2D location $q_2' \in \mathcal{IC}|_{D-1,D}^{q_2|_{[D-2]}}$ which dominates some 2D location,*

[9] By domination, we mean strict domination

Algorithm 3 Multi-D `FrugalSpillBound` (η)

Compilation Phase:
Set: $q_{cur} = (\frac{1}{r}, \cdots, \frac{1}{r})$;
Set: $\beta = \sqrt[D]{\eta}$;
while contours are remaining **do**
  for $q_{cur}.1 = \frac{1}{r}$; $q_{cur}.1 \leq 1$; $q_{cur}.1 = \beta q_{cur}.1$ **do**
    */*Loop for each of the dimensions from* 1 *to* $D-2$ **/*
    for $q_{cur}.(D-2) = \frac{1}{r}$; $q_{cur}.(D-2) \leq 1$; $q_{cur}.(D-2) = \beta q_{cur}.(D-2)$ **do**
      */*Freeze dimensions* $1, \ldots,$ *D-2, vary D and D-1 */*
      $q_{min} = q_{max} = q_{cur}$;
      $q_{max}.(D-1) = q_{max}.D = 1$;
      $q_{min}.(D-1) = q_{min}.D = \frac{1}{r}$;
      /q_{min}/q_{max} are $origin/terminus$ of D, D-1 2D space */*
      if $Cost(q_{min}) \leq \texttt{CC}_\texttt{k}$ and $Cost(q_{max}) \geq \texttt{CC}_\texttt{k}$ **then**
        Call 2D `FSB` with a cost factor of $(\beta)^2$, to cover the $(\beta)^{D-2} \cdot \texttt{CC}_\texttt{k}$ cost 2D contour
      end if
    end for
    */*End of for loops for each of the dimensions from* 1 *to* $D-2$ **/*
  end for
  / Move to next contour */*
end while

Execution Phase:
Run the multi-D `SB` algorithm on the plans corresponding to BCS_i, of every contour $\mathcal{IC}_i$;

$q'_1 \in \mathcal{IC}|^{q_1|_{[D-2]}}_{D-1,D}$. *Since* $q_2|_{[D-2]} \succ q_1|_{[D-2]}$, *we can conclude*

$$\begin{aligned} q_2 &:= q_2|_{[D-2]} \circ q'_2 \\ &\succ q_1 := q_1|_{[D-2]} \circ q'_1 \end{aligned} \tag{10}$$

By PCM, there is a contradiction that q_1, q_2 *belong to the same contour and have same cost.*

Lemma 5.2 *Consider two distinct contours* $\mathcal{IC}_1, \mathcal{IC}_2$ *s.t.* $\texttt{CC}_2 = (\beta)^{D-2} * \texttt{CC}_1$, *and two hyperplanes defined by* $D-2$ *dimensional locations,* q *and* q_β *where* $q_\beta.j = \beta * q.j, \ \forall j \in [D-2]$. *Then, the 2D contour obtained by intersecting* q_β *hyperplane with* $\mathcal{IC}_2$, *covers the one obtained by intersecting* q *hyperplane with* $\mathcal{IC}_1$. *That is, sub-contour* $\mathcal{IC}_{i_2}|^{q_\beta}_{D-1,D}$ *covers* $\mathcal{IC}_{i_1}|^{q}_{D-1,D}$.

Proof 9 *For sake of contradiction, let us say that sub-contour* $\mathcal{IC}_{i_2}|^{q_\beta}_{D-1,D}$ does not *cover* $\mathcal{IC}_{i_1}|^{q}_{D-1,D}$. *Further, this means there exist a 2D location* $q'_1 \in \mathcal{IC}_{i_1}|^{q}_{D-1,D}$ *which dominates some 2D location,* $q'_2 \in \mathcal{IC}_{i_2}|^{q_\beta}_{D-1,D}$. *We know that location* $q' \in \mathcal{IC}_1$, *where* $q' := q \circ q'_1$. *This means that cost of location* q' *is equal to* $\texttt{CC}_1$. *Since* $q'_1 \succ q'_2$, *cost of location* q'_3, *where* $q'_3 := q \circ q'_2$, *is strictly less than* $\texttt{CC}_1$. *By concavity, then the cost of location* q'_β, $q'_\beta := q_\beta \circ q'_2$, *has cost strictly less than* $\texttt{CC}_2$. *This contradicts the fact that* $q'_\beta \in \mathcal{IC}_2$, *and thus having cost equal to* $\texttt{CC}_2$.

Lemma 5.3 *Let* BCS_i *be the bounded contour-covering set output for contour* $\mathcal{IC}_i$ *using Multi-D* `FSB`, *then*

1. *every location in* $\mathcal{IC}_i$ *is dominated by some location in* BCS_i

2. *cost of the dominating location (found in part 1) in BCS_i is at most $\eta \cdot \texttt{CC}_\texttt{i}$*

Proof 10 *1. Consider a location $q \in \mathcal{IC}_i$. The proof goes by eventually constructing a location $dom \in BCS_i$, which dominates q. We know that each of $\{q_{cur}.1, \cdots, q_{cur}.(D-2)\}$ in Algorithm 3, is iteratively increased from $1/r \rightarrow 1$, with a step size of β. Thus, it is easy to see that there exist a q_{cur} such that*

$$\frac{q_{cur}.j}{\beta} < q.j \leq q_{cur}.j \; \forall j \in [D-2]$$

In other words, q_{cur} dominates q when projected on its first $D-2$ dimensions. and thus we set $dom.j = q_{cur}.j \; \forall j \in [D-2]$. Let q_β be D dimensional location such that $q_\beta.j = q_{cur}.j/\beta$. Then, Lemma 5.1 shows that sub-contour $\mathcal{IC}_i|_{D-1,D}^{q_\beta|_{[D-2]}}$ covers $\mathcal{IC}_i|_{D-1,D}^{q|_{[D-2]}}$. Furthermore from Lemma 5.2, we know that sub-contour $\mathcal{IC}_{i_1}|_{D-1,D}^{q_{run}|_{[D-2]}}$ covers $\mathcal{IC}_i|_{D-1,D}^{q_\beta|_{[D-2]}}$ where $\texttt{CC}_{\texttt{i}_1} = (\beta)^{D-2} * \texttt{CC}_\texttt{i}$. From 2D FSB*, we know that every location in $\mathcal{IC}_{i_1}$ is covered by the covering set. This implies that there exist an assignment of values for $(dom.(D-1), dom.(D))$ s.t. dom dominates q.*

2. *Since we are covering the $(\beta)^{D-2} \cdot \texttt{CC}_\texttt{i}$ contour, with an cost inflation factor of at most β^2. Leveraging the proof of Lemma 4.2, we conclude that any location in BCS_i has at most $\eta \cdot \texttt{CC}_\texttt{i}$ cost.*

Next, let us bound the number of optimization calls required by multi-D FSB algorithm per contour.

Lemma 5.4 *The number of optimization calls per contour, for an MSO relaxation of η, is upper bounded by $2 * (D * \log_\eta(r))^{D-1}$*

Proof 11 *We know from Lemma 4.3 that total number of optimization calls for 2D* FSB *is $2 * \log_\beta(r)$. However, the 2D algorithm is executed $(\log_\beta(r))^{D-2}$ times. Equivalently the factor can be rewritten as, $(\frac{\log_2(r)}{\log_2(\sqrt[D]{\eta})})^{D-2}$, or $\{D^{D-2} * (\log_\eta(r))^{D-2}\}$. Thus, optimization calls per contour is upper bounded by $2 * (D * \log_\eta(r))^{D-1}$.*

Theorem 5.5 *The compile-time overheads reduction, γ, of Multi-D* FSB *is at least $r^D/(2 \cdot m \cdot (D \cdot \log_\eta r)^{D-1})$.*

Theorem 5.6 *MSO relaxation of Multi-D* FSB *is at most η.*

Proof 12 *Same as 2D proof.*

6 Experimental Evaluation

In this section, we profile the $\gamma - \eta$ performance of FrugalSpillBound (FSB) on a representative set of complex OLAP queries, using SB's performance as the reference baseline. The experimental framework, which is similar to that of [4], is described first, followed by an analysis of the results.

6.1 Database and System Framework

Our test workload is comprised of **21** complex and representative SPJ queries from the TPC-DS benchmark, operating at the base size of 100 GB.[10] The number of relations in the query suite vary from 4 to 10, and a spectrum of join-graph geometries are modeled, including *chain*, *star*, *branch*, etc. Further, we wish to maximize the range of cost values, and hence the number of effective dimensions and contours in the ESS, in order to create the most challenging environments for robustness. This is achieved through an index-rich physical schema that creates indexes on all the attribute columns appearing in the queries.

With the above setup, the initial number of ESS dimensions, comprising of all filter and join predicates, ranged from 5 to 12 dimensions across the queries. Subsequently, after executing the Dimension Reduction algorithm discussed in Section 5.1, the effective dimensionality came down to a span of 3 to 5 dimensions. As might be expected, the surviving effective dimensions only feature *join predicates*, since the filter predicates were all either accurately estimated by the attribute histograms, or had low MSO impact factors and were therefore eliminated by DimRed.

To succinctly characterize the queries, the nomenclature aD_Qb is employed, where a specifies the number of epps, and b the query number in the TPC-DS benchmark. For example, 3D_Q15 indicates TPC-DS Query 15 with three of its predicates considered to be error-prone.

The database engine used in our experiments is a modified version of the PostgreSQL 9.4 [14] engine, with the primary additions being: (a) *Selectivity Injection*, required to generate the ESS for SB and the bounded contour-covering set for FSB; (b) *Abstract Plan Execution*, required to instruct the execution engine to execute a particular plan; (c) *Plan Recosting*, required to cost an abstract plan to a query; and (d) *Time-limited execution*, required to implement the calibrated sequence of executions with associated time budgets.

6.2 Empirical Validation of APC

We begin with an experimental validation of the Axis-Parallel Concavity (APC) assumption that underlies our entire construction of FSB. For this purpose, we obtained the cost functions of the POSP (Parametric Optimal Set of Plans) plans over the ESS using the selectivity injection feature for all the queries considered in our evaluation. Then, we verified, for each cost function, whether its slope was monotonically non-increasing with selectivity for every 1D projection of the function. Representative results of this evaluation, reflecting 120-plus plans sourced from our suite of benchmark queries, are tabulated in Table 2, for both the constituent PCFs and the aggregate OCS.

In the table, a cell corresponding to OCS (or PCF), under *Average*, captures the % of locations in ESS satisfying the assumption averaged over OCSs (or PCFs) in a query along different projections. Supporting metrics such as *Median*, *Minimum* and *Maximum* are also enumerated to provide a sense of the overall distribution. Note that our FSB approach requires concavity only on the OCS, and the vast majority ($> 95\%$) of locations in the ESS satisfy this slope constraint. Moreover, the median value being 100% for most queries indicates that the majority of OCSs and PCFs do not violate the assumption at all. Further, even the rare violations that surfaced were found to be artifacts of rounding errors, cost-modeling errors, and occasional PCM violations due to the PostgreSQL query optimizer not being entirely cost-based.

[10] From a *conceptual* perspective, the FSB approach is generically applicable to the entire benchmark – however, our current prototype implementation is restricted to SPJ queries, making it infeasible to evaluate the full benchmark at this time.

Table 2: % LOCATIONS IN ESS SATISFYING APC

Query		Average	Median	Min	Max
3D_Q15	OCS	100	100	100	100
	PCF	100	100	100	100
3D_Q96	OCS	100	100	100	100
	PCF	100	100	100	100
4D_Q7	OCS	100	100	100	100
	PCF	98.4	100	74.4	100
4D_Q26	OCS	99.7	100	98.8	100
	PCF	99.7	100	93.9	100
4D_Q27	OCS	100	100	100	100
	PCF	99.2	100	75.2	100
5D_Q19	OCS	100	100	100	100
	PCF	100	100	100	100
5D_Q84	OCS	96.9	96.5	96.5	97.6
	PCF	94.5	96.8	71.2	100
5D_Q91	OCS	100	100	100	100
	PCF	100	100	98.4	100

6.3 Theoretical Characterization of $\gamma - \eta$

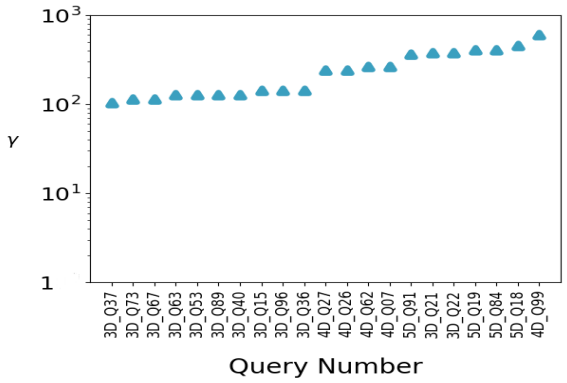


Figure 9: Theoretical Overheads Reduction ($\eta = 2$)

Using the formula derived in Theorem 5.5, we evaluated the γ value for our suite of benchmark queries with η set to 2, and these results are shown in Figure 9 on a *log scale*. We observe here a consistent decrease by more than two orders of magnitude for FSB, i.e. $\gamma >= 100$, over all the queries. Further, and importantly, the decrease shows a trend of being *magnified* with dimensionality. For instance, the overheads decrease by a factor of almost 400 for the five-dimensional 5D_Q84.

6.4 Empirical Characterization of $\gamma - \eta$

We now turn our attention to assessing the *empirical* reduction in compilation overheads achieved by FSB for the above database environment – these results are captured in Figure 10. We see here that for most of the queries, the savings are over *three* orders of magnitude. Furthermore, quite a few of the 4D and 5D queries even reach *four* orders of magnitude reduction – in fact, the overheads saving for 5D_Q91 is by a factor of almost 40000! When the effective dimensionality and the number of contours is moderate, as in the case of the initial three 3D queries in Figure 10, the savings become saturated at

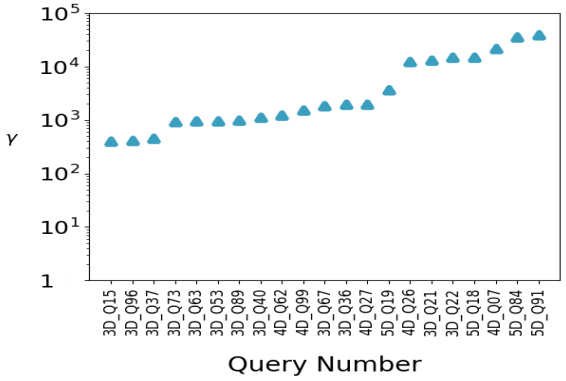


Figure 10: Empirical Overheads Reduction ($\eta = 2$)

around 2.5 orders of magnitude since the overheads reach a low value in absolute terms itself, of the order of a few thousand optimization calls. However, if either the dimensionality and/or the contour density is high, then greater savings become feasible.

The reasons for the considerable gap between the theoretical and empirical values include the following:

- Our conservative formulation in Lemma 3.2 for the distances covered by the forward jumps in FSB. These jumps are based on the slope of the optimal plan function at the corresponding location, but the lengths of the jumps in practice are considerably more due to the concave trajectory. For instance, we found that with 5D_Q84, around 60 percent of the jump lengths exceeded 1.5 times the guaranteed value, while about 20 percent were more than twice the guaranteed value.

- Our conservative assumption that all covering contours start from $1/r$ and work their way upto the maximum selectivity of 1. In practice, however, the contour traversals could be much shorter. As a case in point, we found that with 5D_Q84, around 80 percent of the underlying 2D contour explorations were skipped based on the cost condition check in line 13 of Algorithm 3.

6.5 Validation of MSO Relaxation Property

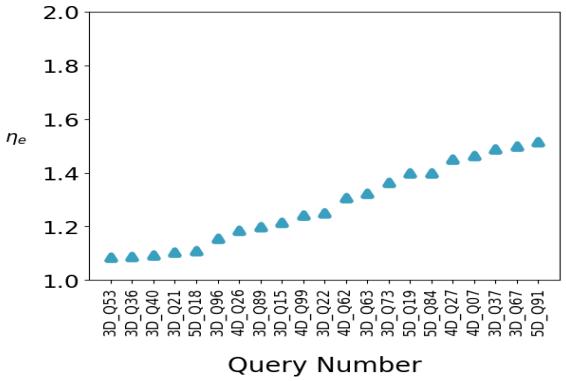


Figure 11: Empirical MSO Ratio ($\eta = 2$)

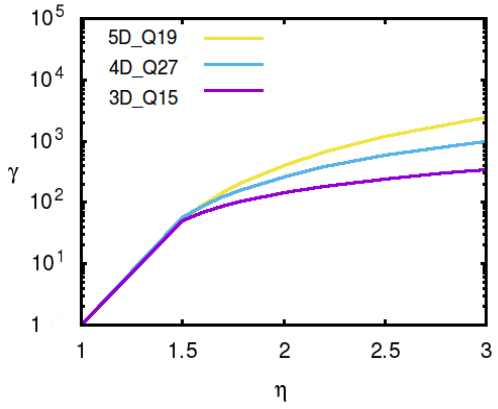


Figure 12: FSB Tradeoff (Theoretical)

A legitimate concern about FSB could be that while it guarantees maintenance of the η constraint in the theoretical framework, the MSO relaxation may exceed η in the *empirical* evaluation. To assess this possibility, we explicitly evaluated the empirical MSO ratio, η_e, incurred by FSB relative to SB. This was accomplished by exhaustively considering each and every location in the ESS to be q_a, and then evaluating the sub-optimalities incurred for these locations by SB and FSB. Finally, the maximum of these values was taken to represent the empirical MSO of each algorithm.

Contrary to our fears, the η_e values of FSB are always within the η $(= 2)$ factor as shown in Figure 11. In fact, the η_e factors are within 1.5 for all queries. The main reason for the low η_e values in practice is due to the aggressive half-space pruning at each contour, and especially at the final contour.

6.6 Dependency of γ on η

Thus far, we have analyzed the FSB results for the specific η setting of 2. We now move on to evaluating the γ behavior for different settings of η. This tradeoff is captured in Figure 12 for η values ranging over $[1, 3]$ for three different queries – Q15, Q27 and Q19 – with ESS dimensionalities of 3, 4 and 5, respectively.

We see here an initial exponential increase in overheads reduction while going from $\eta = 1$ to $\eta = 2$, but this increase subsequently tapers off for larger values of η. For instance, with 3D_Q15, the number of optimization calls decreases steeply from 10^6 to 7010 when η is increased from 1 to 2, and then goes down marginally to 2950 calls when η is further increased to 3. The plateauing of the improvement with increasing η is because a certain minimum number of optimization calls is required for the basic functioning of the FSB algorithm.

6.7 Wall-Clock Time Experiments

All the experiments thus far assessed the $\gamma - \eta$ profile in the abstract world of optimizer cost values. We now present an actual execution experiment, where the end-to-end real-time performance (i.e. wall-clock times) was explicitly measured for the FSB and SB algorithms. Our representative example is based on TPC-DS Q19 featuring 5 error-prone predicates.

As mentioned previously, the task of identifying the contours is inherently amenable to parallelism. Even after exploiting this feature on a 64-core workstation platform, SB took a *few days* to identify all the contours for 5D_Q19. In marked contrast, a parallel version of the BCS identification in FSB, which utilizes the fact that there are $(D * \log_\eta(\frac{1}{\epsilon}))^{D-2}$ independently-explorable $2D$ segments per

contour, completed the identification within *10 minutes* (for $\eta = 2$).

After building the ESS, SB took around 20 mins to complete its query execution incurring a sub-optimality of 4.8. On the other hand, FSB completed in around 26 mins, resulting in a sub-optimality of 6.2. The drilled-down information of plan executions for every contour with FSB can be seen in Table 3.

Table 3: FSB EXECUTION ON TPC-DS QUERY 19

Contour no.	e_1	e_2	e_3	e_4	e_5	Time (sec.)
1	-	p_1	-	p_2	p_3 (**100**)	54.1
2	-	p_4	-	p_5	-	122.5
3	-	p_4 (**1**)	-	-	-	182.1
3	-	-	-	p_5	-	251.4
4	-	-	-	p_5	-	357.8
5	-	-	-	p_5	-	509.9
6	p_6 (**5**)	-	-	-	-	789.2
6	-	-	p_7 (**96**)	-	-	1051.6
7	-	-	-	p_8 (**99**)	-	1562

So, overall, SB took days to create the ESS and execute this instance of Q19, whereas FSB required only (10 minutes $+$ 26 minutes) $=$ 36 minutes to complete the entire query processing. This means that even if the supposedly ad-hoc query eventually turns out to be a canned query, it would take more than 500 successive invocations before SB begins to outperform FSB.

We conducted additional experiments to establish the practicality of the FSB approach. Specifically, on a representative set of queries, we profiled FSB for its memory usage, CPU usage, and end-to-end latency. The memory usage is also a function of the server's database configuration, which was set with the PostgreSQL tuning tool [13]. The results, presented in Table 4, demonstrate that FSB's resource requirements are reasonable and easily justified by the substantive performance benefits it delivers.

Table 4: RESOURCE USAGE (100 GB)

Query	Memory Usage (MB)	CPU Times (mins)	Latency (mins)
3D_Q15	360	1.4	28.1
3D_Q96	220	1.3	17.8
4D_Q7	489	1.2	23
4D_Q26	490	1.5	12.6
4D_Q27	464	1.8	30.5
5D_Q19	1000	11	36
5D_Q84	348	2.8	10.1
5D_Q91	828	1.3	4.3

7 Deployment Aspects

Over the preceding sections, we have conducted a theoretical characterization and empirical evaluation of the `FrugalSpillBound` algorithm. We now discuss some pragmatic aspects of its usage in real-world contexts. Most of these issues have already been previously discussed in [10], in the context of the `SpillBound` algorithm, and we therefore only summarize the salient points here for easy reference.

First, we have implicitly assumed a perfect cost model in our study, but this is rarely the case in practice. However, if we were to be assured that the cost modeling errors, while non-zero, are *bounded* within a δ error factor, then the MSO guarantees in this paper will carry through modulo an inflation by a factor of $(1+\delta)^2$. For instance, $\delta = 0.3$ is reported in [1].

Second, it is important to note that `SpillBound` and `FrugalSpillBound` are *not* substitutes for conventional query optimizers, but are intended to complementarily co-exist with the traditional setup. We currently leave it to the user's discretion about the specific approach to employ for a given query instance – however, we have also begun exploring the use of machine learning techniques to make this determination.

Finally, both `SpillBound` and `FrugalSpillBound` are *intrusive* in that they require changes in the core engine to support the various functionalities such as plan spilling and monitoring of operator selectivities. However, our experience with PostgreSQL is that these facilities can be incorporated relatively easily. As an aside, the BCS approach that we presented here to achieve a tradeoff between guarantees and overheads, can also be used in conjunction with the original `PlanBouquet` algorithm, which operates purely with API functionality.

8 Related Work

In the prior robust query processing literature, as discussed below, there have been two strands of work – the first delivering savings on optimization overheads, and the other primarily addressing the query execution performance. In this context, `FrugalSpillBound` appears a unique proposition since it offers an attractive *tradeoff* between these two competing and complementary aspects.

Compilation Overheads The primary work in this area has been in the context of Parametric Query Optimization (PQO), where the objective is to have *precomputed* the appropriate plans for freshly submitted queries. In [9], the selectivity space was decomposed into polytopes that approximate plan-optimality regions, based on the geometric heuristic that "If all vertices of a polytope have the same optimal plan, then the plan is also optimal *within* the entire polytope". However, this assumption, as well as the presence of regular boundaries for the optimality regions, were later shown in [7] to be largely violated in industrial-strength settings.

Instead of trying to characterize the entire selectivity space in advance, [3] took an alternative approach of reducing the PQO overheads by restricting attention to the query workload that is actually submitted to the system, and thereby progressively and efficiently explore the parameter space. In our setting, however, since we are apriori unaware of the query location, the BCS is constructed in a manner that is agnostic to this location.

More recently, [5] identified a geometric property, referred to as Bounded Cost Growth (BCG), that typically holds on plan cost functions. It leveraged this property to ensure bounded sub-optimalities of the PQO choices, relative to the ideal plan at the query location. In BCG, the relative increase of plan costs is modeled as a low-order polynomial function of the relative increase in plan selectivities. In

fact, using the identity function for this polynomial is itself found to be generally satisfactory. Our use of concavity is similar to BCG in that, when the polynomial is the identity function, it is shown below that any PCF that satisfies APC also satisfies BCG.

8.1 BCG

Let us now see BCG's definition formally. For any PCF $\mathcal{F}_p$:

$$\mathcal{F}_p(\alpha * q.j) \leq f(\alpha) * \mathcal{F}_p(q.j) \;\; \forall j \in \{1, \ldots, D\}, \forall \alpha \geq 1 \tag{11}$$

where $f(\alpha)$ is an increasing function and $\mathcal{F}_p(q)$ represents the cost of the corresponding plan at location q. In words, for any plan, if the selectivity is increased in any one of the dimensions by a factor $\alpha \geq 1$, then the cost of the plan also increases by a factor at most $f(\alpha)$. Moreover, they also claim that $f(\alpha) = \alpha$ would suffice in practice. As in the case of our axis-parallel concave assumption, they also show that if BCG holds true for POSP plans then it is also true for PIC.

8.2 Concavity implies BCG

Let us consider a PCF $\mathcal{F}_p$, if $\mathcal{F}_p$ is axis-parallel concave, then we will show that the PCF also satisfies the BCG assumption when $f(\alpha) = \alpha$. For this, we just need to show the implication over an 1D projection, which then easily generalizes for the generic scenario. Consider a location q, whose projection on dimension j (i.e., $q.j$) has slope m on $\mathcal{F}_p$. Thus the tangent at $q.j$ can be expressed as a line of the form $\mathcal{F}_p(q.j) = m * q.j + c$. However, $c \geq 0$ for the function value to be non-negative for $q.j = 0$. Hence, $\mathcal{F}_p(\alpha * q.j) \leq m(\alpha * q.j) + c \leq \alpha * (mq.j + c) = \alpha * \mathcal{F}_p(q.j)$,

Query Execution The `PlanBouquet` approach [4], based on selectivity discovery instead of estimation, provided for the first time, guarantees on the worst-case execution performance. However, its bounds were a function of not only the query, but also the optimizer's behavioral profile over the underlying database platform. `SpillBound` materially extended `PlanBouquet` by providing platform-independent guarantees through incorporating plan spilling and selectivity monitoring mechanisms in the database engine. Moreover, its empirical performance was markedly superior to `PlanBouquet`, thanks to an aggressive half-space pruning of the `ESS` in the discovery process, and bounding the number of plan executions required to move from one contour to the next.

Both `PlanBouquet` and `SpillBound` fall under the umbrella of plan-switching approaches. They may therefore appear very similar to run-time heuristic re-optimization techniques such as POP [12] and Rio [2]. A detailed comparison to POP, Rio and related literature wrt selectivity estimation issues was provided in [4]. The key difference is in the provision of performance guarantees. Further, the heuristic techniques use the optimizers plan choice as the starting point, and reoptimize at run-time if the estimates are found to be significantly in error. In contrast, `PlanBouquet` and `SpillBound` always start executing plans from the origin of the selectivity space, ensuring both repeatability of the query execution strategy as well as controlled switching overheads.

9 Conclusions

The recently proposed `SpillBound` and `PlanBouquet` query processing techniques incorporate a selectivity discovery process as opposed to the traditional estimation approach. This leads to the much-needed MSO performance guarantees, an essential ingredient of robust query processing. On

the downside, however, these techniques are suitable only for canned queries due to the enormous compilation overheads that are required before query execution can be initiated.

In this paper, we address the above limitation by designing `FrugalSpillBound` whose compilation overheads are exponentially lower than those of `SpillBound`. Our construction of `FrugalSpillBound` is based on two basic principles: (a) leveraging the axis-parallel concave behavior exhibited by the PCFs and OCS with respect to predicate selectivities in the `ESS`; (b) substituting the original isocost contours with contour-covering sets that are much smaller in size but whose costs are controlled to within a bounded factor of the original contour.

Our theoretical analysis demonstrates that the $\eta - \gamma$ tradeoff is extremely attractive, delivering exponential improvements in γ for linear relaxations in η. Further, the empirical improvements, evaluated on TPC-DS queries, are even higher, by more than an order of magnitude. So, in an overall sense, `FrugalSpillBound` takes an important step towards successfully extending the benefits of MSO guarantees to ad-hoc queries.

In our future work, we intend to investigate the existence of alternative geometric constraints on cost function behavior – for example, a bounded rate of change – and how to leverage such constraints for improving the MSO guarantees and/or compilation overheads.

References

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

[2] S. Babu, P. Bizarro, and D. DeWitt. Proactive re-optimization. In *ACM SIGMOD Conf.*, 2005.

[3] P. Bizarro, N. Bruno, and D. DeWitt. Progressive parametric query optimization. *IEEE TKDE*, 21(4), 2009.

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

[5] A. Dutt, V. Narasayya, and S. Chaudhuri. Leveraging re-costing for online optimization of parameterized queries with guarantees. In *ACM SIGMOD Conf.*, 2017.

[6] G. Goetz. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, 18(3), 1995.

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

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

[9] A. Hulgeri and S. Sudarshan. Anipqo: Almost non-intrusive parametric query optimization for nonlinear cost functions. In *VLDB Conf.*, 2003.

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

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

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

[13] PGTune. https://pgtune.leopard.in.ua/.

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

[15] S. Purandare. Dimensionality reduction techniques for bouquet-based approaches. Master's Thesis, Database System Lab, IISc, 2018, http://dsl.cds.iisc.ac.in/publications/thesis/sanket.pdf.

[16] 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.