Conditional Selectivity for Statistics on Query Expressions

Nicolas Bruno
Microsoft Research
nicolasb@microsoft.com

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

ABSTRACT

Cardinality estimation during query optimization relies on simplifying assumptions that usually do not hold in practice. To diminish the impact of inaccurate estimates during optimization, statistics on query expressions (SITs) have been previously proposed. These statistics help directly model the distribution of tuples on query sub-plans. Past work in statistics on query expressions has exploited view matching technology to harness their benefits. In this paper we argue against such an approach as it overlooks significant opportunities for improvement in cardinality estimations. We then introduce a framework to reason with SITs based on the notion of conditional selectivity. We present a dynamic programming algorithm to efficiently find the most accurate selectivity estimation for given queries, and discuss how such an approach can be incorporated into existing optimizers with a small number of changes. Finally, we demonstrate experimentally that our technique results in superior cardinality estimations than previous approaches with very little overhead.

1. INTRODUCTION

Relational query optimizers explore a large number of execution plans to evaluate an input query, and choose the most efficient alternative in a cost-based manner. The cost estimation for a plan depends on several factors, including resource availability, the specific operators composing the plan, and the size of intermediate results that would be generated during the plan execution. Among these factors, the intermediate-result size (or cardinality) estimation is one of the key sources of inaccuracies during optimization. To estimate cardinality values, the optimizer relies on several simplifying assumptions (e.g., independence between predicates), which often do not hold. Thus, resulting cardinality estimates can be off by orders of magnitude and lead the optimizer to choose low-quality execution plans.

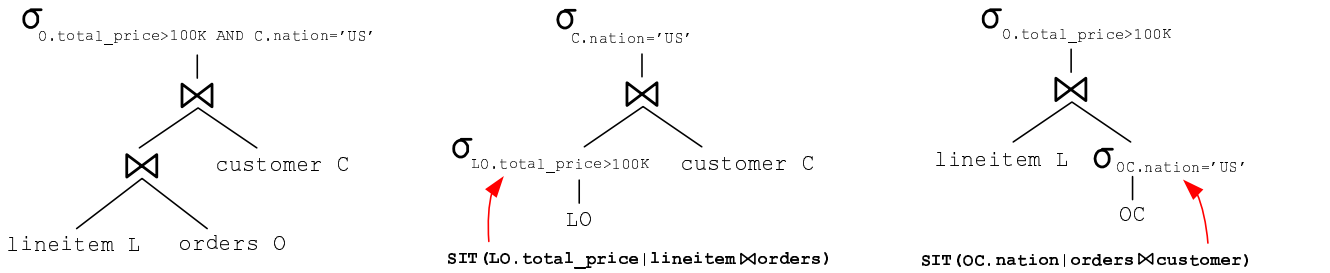
To diminish error propagation through complex query plans, the concept of statistics on query expressions (or on views) was introduced in [4, 26]. Borrowing the terminology from [4], in this paper we refer to these statistics as "SITs". SITs have the same structure and functionality as base-table statistics. However, unlike base-table statistics, SITs are created on the result of executing some query expression. (Note that although in this paper we focus on SITs as histograms, the same ideas can be applied to other statistical estimators, such as wavelets or samples). SITs can then be used to directly model the distribution of tuples on *intermediate* nodes of query execution plans, avoiding inaccurate extrapolation of base-table histograms. The benefits of using SITs during optimization are analogous to those of using materialized views for query execution: materialized views speed up input queries by pre-computing and caching sub-query results; SITs likewise increase the *accuracy* of cardinality estimates of input queries by pre-computing and modelling the *distribution* of intermediate results.

Consider the TPC-H query in Figure 1(a). To obtain its cardinality, a traditional optimizer proceeds as follows: (i) estimates the selectivity of `nation='USA'` and `total_price>100K` using histograms over `customer.nation` and `orders.total_price`, respectively, (ii) multiplies these selectivity values together assuming independence, (iii) estimates the cardinality of the three-way join `lineitem` $\bowtie$ `orders` $\bowtie$ `customer`, and (iv) scales down the join cardinality by the combined selectivity of the filter predicates (again assuming independence). Now suppose that our TPC-H database is skewed. Specifically, the number of line-items for a given order follows a Zipfian distribution. In such a case, although the fraction (or selectivity) of expensive orders with `total_price > 100K` is very small, the fraction of line-items that match with such orders can be very large (with respect to the whole join). The reason is that expensive orders typically consist of many line-items. For that reason, the original cardinality estimation can be a severe underestimate. Intuitively, the problem arises from assuming independence between predicates `total_price>100K` and `lineitem` $\bowtie$ `orders`. Similarly, if the majority of customers live in the US, the same problem also arises from predicate `nation` $=$ `'USA'`.

Now suppose that we create a SIT for `total_price` on the result of evaluating `LO` $=$ `lineitem` $\bowtie$ `orders` (we denote such SIT as SIT(`LO.total_price`|`lineitem` $\bowtie$ `orders`)). Then, using techniques in [4], we rewrite the original query of Figure 1(a) into the equivalent query of Figure 1(b), which exploits the SIT. (Reference [4] identifies whether or not a SIT is applicable to a given plan leveraging materialized view matching techniques.) We now estimate the query cardinality using the SIT instead of the base-table histogram over `orders.total_price`, obtaining a more accurate estimate (the SIT correctly models the interaction between expensive orders and the number of line items that join with them). Similarly, if we create SIT(`OC.nation`|`orders` $\bowtie$ `customer`), we can rewrite the original query of Figure 1(a) into the equivalent query of Figure 1(c) to obtain, again, a better cardinality estimate.

If in our sample database *both* filter predicates are skewed, the cardinality obtained when using any one of the available SITs in

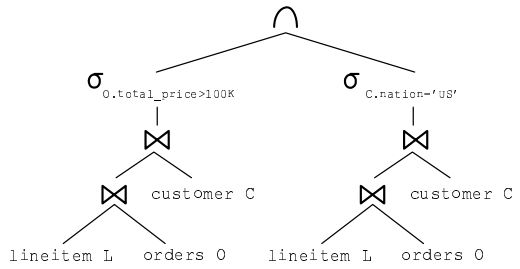
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD 2004 June 13-18, 2004, Paris, France.
Copyright 2004 ACM 1-58113-859-8/04/06 . . . $5.00.



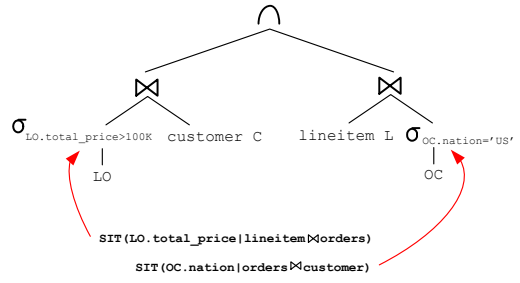
(a) Original Query. (b) Using SIT on `lineitem` ⋈ `orders`. (c) Using SIT on `orders` ⋈ `customer`.

Figure 1: Exploiting SITs using view matching.

isolation can still be an arbitrary underestimation (although more accurate than if no SITs are used). In general, if two or more SITs are available, it is a good idea to use them together (whenever possible) to estimate cardinality values: each SIT might correct errors introduced in different parts of the input query. Unfortunately, in Figure 1 there is no transformation, based purely on view matching techniques, that exploits both SITs *simultaneously*. The reason is that the SITs query expressions are mutually exclusive from a view matching perspective.



(a) Rewriting the original query.



(b) Using both SITs simultaneously.

Figure 2: Exploiting SITs beyond view matching.

Alternatively, consider the following approach: if we were to rewrite the original query of Figure 1(a) as in Figure 2(a), then we could exploit SITs as explained earlier for each sub-query of the intersection operator (see Figure 2(b)) and obtain the final cardinality estimate by assuming independence only at the root node. By using both SITs simultaneously, we eliminate one independence assumption compared to the alternatives in Figures 1(b) and 1(c). Of course, this example relies on a special rewriting that is typically not explored by optimizers. However, it shows that purely relying on the existing view matching infrastructure provided by the optimizer falls short of capturing the whole spectrum of alternatives to exploit SITs for cardinality estimation. It also shows that the best cardinality estimate that can be obtained using previous techniques (e.g., [4]) might be significantly more inaccurate than what is possible to accomplish using the available information. What we need instead is a general framework to decide how to best (and correctly) use available SITs together to estimate cardinality values.

In this paper we focus on the class of `SPJ` queries (see [3] for extensions that handle optional `Group-By` clauses) and introduce a systematic and comprehensive approach to decide how to exploit multiple (possibly conflicting) SITs to estimate cardinality values. As we show in Section 5, our technique results in estimation errors that can be an order of magnitude smaller than those obtained by previous approaches for the same set of available SITs.

As shown in Figures 1 and 2, there are in general different alternatives to exploit available SITs for cardinality estimation. Since our goal is to obtain accurate cardinality estimations, we need a procedure to "rank" different alternatives based on their expected accuracy (we address this issue in Sections 3.2 and 3.5). Unfortunately, as we show in Section 2, there is a combinatorial explosion of alternatives to approximate cardinality values for given SITs. To address this problem, we study some properties that can significantly prune the search space without missing the most accurate alternatives, and introduce an efficient, dynamic-programming based algorithm to enumerate this smaller search space. Since cardinality estimation is an integral aspect of query optimization, the proposed techniques should be aware and exploit certain properties of the optimization process. For instance, during the optimization of a single query, many selectivity requests would refer to "similar" query plans (e.g., it is likely that before requesting an estimation for the plan of Figure 1(a), the optimizer already requested selectivity estimates for many of its sub-plans). It is then important that our technique reuses previous computations during the optimization of the same query to satisfy new selectivity requests.

In this paper we address the challenges discussed above by introducing the concept of *conditional selectivity*. We develop a framework that, given a set of SITs, allows us to easily identify and efficiently search the *full* space of alternatives to obtain accurate estimates by exploiting SITs. We can draw an analogy between the use of SITs and work in traditional query optimization. During optimization, algebraic properties (e.g., associativity and commutativity of joins) provide different equivalent logical plans to represent the same input query. In this paper, conditional selectivity properties result in many alternative ways (we call them *decompositions*) to estimate the same selectivity value. Optimizers efficiently navigate through these equivalent logical plans guided by a cost model, and choose a set of access paths (e.g., sequential or index scans) to implement each logical plan. Analogously, we suggest a dynamic programming algorithm, *getSelectivity*, that exploits the structure of the input query to efficiently search the space of decompositions, and identifies the "best" way to approximate a selectivity value using SITs. In particular, SITs are the implementation mechanism to approximate decompositions, and the search in *getSelectivity* is guided by the expected benefit of each alternative.

The rest of the paper is structured as follows. In Section 2 we introduce conditional selectivity. In Section 3 we present a dynamic programming algorithm to find the most accurate cardinality estimation using SITs. In Section 4 we show how the algorithm can be integrated with relational optimizers. We report experimental results in Section 5 and review related work in Section 6.

2. CONDITIONAL SELECTIVITY

In the previous section we illustrated how relying purely on view matching techniques to exploit SITs can sometimes hinder accurate cardinality estimations. We now introduce the concept of *conditional selectivity* and then discuss how it relates to SITs. Conditional selectivity allows expressing the selectivity of a given SPJ query in many different but equivalent ways. In this paper, we represent an arbitrary SPJ query in a *canonical form* by first forming the cartesian product of the tables referenced in the query, then applying all predicates (including joins) to such cartesian product, and finally projecting out the desired attributes. Thus, we represent a generic SPJ query as $\pi_{a_1,\ldots,a_{n_a}}(\sigma_{p_1\wedge\ldots\wedge p_{n_p}}(R_1\times\ldots\times R_n))$ where a_i are attributes of $R_1\times\ldots\times R_n$, and p_i are predicates over $R_1\times\ldots\times R_n$ (e.g., $R_1.a\leq 25$, or $R_1.x$=$R_2.y$). Each set of predicates $\{p_i\}$ applied to $R_1\times\ldots\times R_n$ results in the subset of tuples that *simultaneously* satisfy all p_i. To estimate the size of the output, or its cardinality, we first approximate the fraction of tuples in $R_1\times\ldots\times R_n$ that simultaneously satisfy all predicates p_i (i.e., the *selectivity* of all p_i), and then multiply such fraction by $|R_1\times\ldots\times R_n|$, which can be obtained by simple lookups over the system catalogs. We now extend the definition of selectivity.

DEFINITION 1. *Let $\mathcal{R}=\{R_1,\ldots,R_n\}$ be a set of tables, and $P=\{p_1,\ldots,p_j\}$ and $Q=\{q_1,\ldots,q_k\}$ be sets of predicates over $R_1\times\ldots\times R_n$. The* conditional selectivity *$\mathcal{S}el_{\mathcal{R}}(P|Q)$ is defined as the fraction of tuples in $\sigma_{q_1\wedge\ldots\wedge q_k}(R_1\times\ldots\times R_n)$ that simultaneously satisfy all predicates in P. Therefore,*

$$\mathcal{S}el_{\mathcal{R}}(P|Q)=\frac{\left|\sigma_{p_1\wedge\ldots\wedge p_j}(\sigma_{q_1\wedge\ldots\wedge q_k}(R_1\times\ldots\times R_n))\right|}{|\sigma_{q_1\wedge\ldots\wedge q_k}(R_1\times\ldots\times R_n)|}$$

If $Q=\emptyset$ we write $\mathcal{S}el_{\mathcal{R}}(P)$, which coincides with the original definition of selectivity.

We denote the set of tables referenced by a set of predicates P as *tables*(P), and the set of attributes mentioned in P as *attr*(P). To simplify the presentation, we also use "P,Q" to denote "$P\cup Q$" and "p,Q" to denote "$\{p\}\cup Q$", where p is a predicate and P and Q are sets of predicates. Finally, for a set of tables $\mathcal{R}=\{R_1,\ldots,R_n\}$, we use $\mathcal{R}^\times$ to refer to the cartesian product $R_1\times\ldots\times R_n$.

In general, for a given query $\sigma_{p_1\wedge\ldots\wedge p_k}(\mathcal{R}^\times)$, our task is to estimate $\mathcal{S}el_{\mathcal{R}}(p_1,\ldots,p_k)$. The key property of conditional selectivity is *atomic decomposition*, which uses the notion of conditional probability to unfold a selectivity value as the product of two conditional selectivity values.

PROPERTY 1 (ATOMIC DECOMPOSITION). *Given a set of tables $\mathcal{R}$ and sets of predicates P and Q:*

$$\mathcal{S}el_{\mathcal{R}}(P,Q)=\mathcal{S}el_{\mathcal{R}}(P|Q)\cdot\mathcal{S}el_{\mathcal{R}}(Q)$$

This property holds for arbitrary sets of predicates and tables, without relying on *any simplifying assumption*. An atomic decomposition divides the problem of estimating $\mathcal{S}el_{\mathcal{R}}(P,Q)$ into two sub-problems (estimating $\mathcal{S}el_{\mathcal{R}}(P|Q)$ and $\mathcal{S}el_{\mathcal{R}}(Q)$). The decomposition property is essential to develop a framework for exploiting SITs. In fact, we will rely on available SITs[1] to estimate the first factor $\mathcal{S}el_{\mathcal{R}}(P|Q)$. For instance, we can use SIT$(R.a|R\bowtie S)$ to approximate $\mathcal{S}el_{\{R,S\}}(R.a<10|R\bowtie S)$. In turn, if Q consists of a single predicate we use a standard technique to estimate the second factor $\mathcal{S}el_{\mathcal{R}}(Q)$, or otherwise recursively apply another atomic decomposition to $\mathcal{S}el_{\mathcal{R}}(Q)$.

By repeatedly applying atomic decompositions, we might obtain a very large number of alternative expressions for a given selectivity value, which we simply call *decompositions*. A decomposition of a given selectivity value is then an expression of the form $\mathcal{S}_1\cdot\ldots\cdot\mathcal{S}_k$, where each $\mathcal{S}_i=\mathcal{S}el_{\mathcal{R}_i}(P_i|Q_i)$ and $Q_k=\emptyset$. If each factor $\mathcal{S}el_{\mathcal{R}_i}(P_i|Q_i)$ is calculated precisely, then every possible decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ evaluates to the same value (because we obtain each alternative decomposition through a series of equalities). However, in real scenarios only a small number of SITs are available. It follows that, depending on the set of SITs at hand, some decompositions might be more accurate than others. Suppose that we assign to each decomposition of a selectivity value $\mathcal{S}el_{\mathcal{R}}(P)$ a measure of how accurately the decomposition can be approximated using the current set of available SITs. Then, estimating $\mathcal{S}el_{\mathcal{R}}(P)$ can be seen as an optimization problem: we want to obtain the "most accurate" decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ for the given set of available SITs (see Section 3.2 for our definition of accuracy). In principle, we could explore exhaustively all possible decompositions of $\mathcal{S}el_{\mathcal{R}}(P)$, estimate the accuracy of each decomposition and return the most accurate one. Unfortunately, this approach is prohibitively expensive given the large number of possibilities, as illustrated below.

LEMMA 1. *The number of possible decompositions of $\mathcal{S}el_{\mathcal{R}}(p_1,\ldots,p_n)$, denoted $T(n)$, is bounded by*

$$0.5\cdot(n+1)!\leq T(n)\leq 1.5^n\cdot n!,\qquad \textit{for all } n\geq 1$$

In the next section we introduce a comprehensive approach to obtain the most accurate cardinality estimation for a given query and a set of available SITs. We discuss how the *accuracy* of a decomposition is measured and derive a dynamic programming algorithm that efficiently returns the most accurate decomposition of a given selectivity value.

3. OBTAINING THE BEST SELECTIVITY ESTIMATION

Conditional selectivity provides a framework for exploiting SITs to obtain query cardinality estimates. Unfortunately, the space of decompositions of a given selectivity value can be very large. In Section 3.1 we discuss how to safely prune this search space (without missing the most accurate decomposition) by leveraging some properties of conditional selectivity values. Then, in Section 3.2 we discuss how to estimate the accuracy of a given decomposition (which in turn allows us to rank different decompositions). Next, in Section 3.3 we show how to exploit available SITs to approximate a single factor in a decomposition. In Section 3.4 we derive an dynamic-programming algorithm that returns the most accurate decomposition for a selectivity value (we also show that the complexity of the algorithm is drastically reduced compared to the approach that enumerates all alternative decompositions). Finally, in Section 3.5 we introduce an improved procedure to rank alternative decompositions that results in better estimations than the simple metric described in [4] (see Section 5 for experimental results).

3.1 Safely Pruning Decompositions

In this section we introduce the notion of "separability" of a decomposition. The *separability* property can be seen as a syntactic

[1]Determining which SITs to use (and how to use them) to approximate a selectivity factor is discussed in Section 3.3.

notion of independence that allows simplifying a selectivity value whenever certain properties hold, and can further reduce the search space without missing any relevant decomposition.

DEFINITION 2. *We say that $\mathcal{S}el_{\mathcal{R}}(P|Q)$ is* separable *(with Q possibly empty) if we can find non-empty sets X_1 and X_2 such that $P \cup Q$=$X_1 \cup X_2$ and tables$(X_1) \cap$ tables(X_2)=$\emptyset$. We also say that X_1 and X_2* separate *$\mathcal{S}el_{\mathcal{R}}(P|Q)$.*

Intuitively, the expression $\mathcal{S}el_{\mathcal{R}}(P|Q)$ is separable if $\sigma_{P \wedge Q}(\mathcal{R}^{\times})$ combines some tables in $\mathcal{R}$ by using cartesian products. Note, however, that even if the original query does not use any cartesian product, after applying atomic decompositions some factor might become separable. Consider the non-separable expression $\mathcal{S}el_{\{R,\,S\}}(R.a{<}10, S.b{>}5, R.x{=}S.y)$. After applying an atomic decomposition, we get $\mathcal{S}el_{\{R,\,S\}}(R.x{=}S.y|R.a < 10, S.b > 5) \cdot \mathcal{S}el_{\{R,\,S\}}(R.a < 10, S.b > 5)$, whose second factor is separable. We now introduce the separable decomposition property.

PROPERTY 2 (SEPARABLE DECOMPOSITION). *Suppose that $\{P_1, P_2\}$ and $\{Q_1, Q_2\}$ are partitions of P and Q, and X_1=$P_1 \cup Q_1$ and $X_2 = P_2 \cup Q_2$ separate $\mathcal{S}el_{\mathcal{R}}(P|Q)$. Let $\mathcal{R}_1$=tables(X_1) and $\mathcal{R}_2$=tables(X_2). Then,*

$$\mathcal{S}el_{\mathcal{R}}(P|Q) = \mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1) \cdot \mathcal{S}el_{\mathcal{R}_2}(P_2|Q_2)$$

EXAMPLE 1. *Since $\{T.b = 5\}$ and $\{R.x{=}S.y, S.a < 10\}$ separate $s = \mathcal{S}el_{\{R,S,T\}}(T.b{=}5, S.a < 10|R.x{=}S.y)$, we rewrite s=$\mathcal{S}el_{\{R,S\}}(S.a < 10|R.x{=}S.y) \cdot \mathcal{S}el_{\{T\}}(T.b > 5)$. Here, both resulting factors are no longer separable.* ∎

Using the separable decomposition property, we make the following natural assumption concerning histograms.

ASSUMPTION 1 (MINIMALITY OF HISTOGRAMS). *Suppose that $\mathcal{S}el_{\mathcal{R}}(P|Q)$ is separable as $\mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1) \cdot \mathcal{S}el_{\mathcal{R}_2}(P_2|Q_2)$, and let H be a histogram that directly approximates $\mathcal{S}el_{\mathcal{R}}(P|Q)$. We assume that there exist histograms H_1 and H_2 approximating $\mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1)$ and $\mathcal{S}el_{\mathcal{R}_2}(P_2|Q_2)$, respectively, such that: (i) H_1 and H_2 combined require no more space than H, and (ii) approximating $\mathcal{S}el_{\mathcal{R}}(P|Q)$ with H_1 and H_2 is as accurate as with H.*

Consider, for instance, $\mathcal{S}el_{\{R,\,S\}}(R.a < 10, S.b > 20)$, which is separable as $\mathcal{S}el_{\{R\}}(R.a < 10) \cdot \mathcal{S}el_{\{S\}}(S.b > 20)$. Suppose that we estimate each factor using a unidimensional histogram on $H(R.a)$ and $H(S.b)$, respectively, and then multiply the resulting selectivity values assuming independence (which holds in this case). we claim that this approach is at least as accurate as directly using a two-dimensional histogram $H(R.a, S.b)$ built on $R \times S$. Since the independence assumption holds, the joint distribution over $\{R.a, S.b\}$ can be accurately modelled from unidimensional distributions over $R.a$ and $S.b$.

We can then avoid maintaining SITs that directly approximate separable factors of decompositions, because such SITs can be replaced by more accurate and space-efficient ones. For that reason, we can safely discard from the search space all decompositions $\mathcal{S} = \mathcal{S}_1 \cdot \ldots \cdot \mathcal{S}_n$ for which some $\mathcal{S}_i$ is separable, without missing the most accurate decomposition. The separable decomposition property and the minimality assumption for histograms can substantially reduce the search space, since we avoid considering a large number of decompositions.

In general, there is always a unique decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ into non-separable factors of the form $\mathcal{S}el_{\mathcal{R}_i}(P_i)$. That is, if we start with $\mathcal{S}el_{\mathcal{R}}(P)$ and repeatedly apply separable decompositions until no single resulting factor is separable, we always obtain the same non-separable decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$.

LEMMA 2. *There is a unique decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ as $\mathcal{S}el_{\mathcal{R}_1}(P_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_n}(P_n)$, where each $\mathcal{S}el_{\mathcal{R}_i}(P_i)$ is not separable. We call it the* standard decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$.

We will use this result in the Section 3.4 to safely prune the search space of decompositions.

3.2 Ranking Candidate Decompositions

We now define the notion of *error*, which measures the (estimated) accuracy of a decomposition for given SITs.

DEFINITION 3. *Let $s = \mathcal{S}el_{\mathcal{R}}(p_1, \ldots, p_n)$ be a selectivity value, and $\mathcal{S} = \mathcal{S}_1 \cdot \ldots \cdot \mathcal{S}_k$ be a decomposition of s, where $\mathcal{S}_i = \mathcal{S}el_{\mathcal{R}_i}(P_i|Q_i)$. If we use SIT H_i to approximate $\mathcal{S}_i$, then error$(H_i, \mathcal{S}_i)$ measures the accuracy of H_i approximating $\mathcal{S}_i$. The value error$(H_i, \mathcal{S}_i)$ is a positive real number, where smaller values represent better accuracy. The (estimated) overall error for $\mathcal{S}$=$\mathcal{S}_1 \cdot \ldots \cdot \mathcal{S}_k$ is given by an aggregate function $E(e_1, \ldots, e_n)$, where e_i = error$(H_i, \mathcal{S}_i)$. We focus on* monotonic *and* algebraic *aggregate functions:*

- *Function E is* monotonic *if every time that $x_i \leq x'_i$ for all i, we have that $E(x_1, \ldots, x_n) \leq E(x'_1, \ldots, x'_n)$. Monotonicity is a reasonable property for functions measuring overall accuracy [6, 9, 10]: if each error e'_i is at least as high as e_i, then the overall error $E(e'_1, \ldots, e'_n)$ should be at least as high as $E(e_1, \ldots, e_n)$.*
- *Function F is* distributive *if, for some function G, it holds that $F(x_1, \ldots, x_n)$= $G(F(x_1, \ldots, x_i), F(x_{i+1}, \ldots, x_n))$. For instance,* `max` *(with G=*`max`*) and* `count` *(with G=*`sum`*) are distributive functions. A function E is* algebraic *if there is a function H and distributive functions $F_1, \ldots, F_m$ such that $E(x_1, \ldots, x_n)$=$H(F_1(x_1, \ldots, x_n), \ldots, F_m(x_1, \ldots, x_n))$. Function* average *is algebraic (with F_1=*`sum`*, F_2=*`count`*, and $H(x, y) = x/y$). To simplify the notation, we define $E_{merge}(E(x_1, \ldots, x_i), E(x_{i+1}, \ldots, x_n))$=$E(x_1, \ldots, x_n)$ for an algebraic function E. Therefore, as an example, we have that* $\texttt{avg}_{merge}(\texttt{avg}(1,2), \texttt{avg}(3,4))$=$\texttt{avg}(1,2,3,4)$.

Monotonicity imposes the principle of optimality for error values (i.e., the components of a globally optimal solution are themselves optimal), and allows a dynamic programming strategy to find the best decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$. That is, we can find the most accurate decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ by trying all atomic decompositions $\mathcal{S}el_{\mathcal{R}}(P)$=$\mathcal{S}el_{\mathcal{R}}(P_1|P_2) \cdot \mathcal{S}el_{\mathcal{R}}(P_2)$, recursively obtaining the optimal decomposition of $\mathcal{S}el_{\mathcal{R}}(P_2)$ (only once), and combining the partial results. The key property of algebraic aggregates is that a small fixed-size vector can summarize all necessary subaggregations and therefore bounds the amount of information to carry over between recursive calls to obtain *error* values.

Below we adapt a simple error function introduced previously in [4], and then in Section 3.5 we introduce a novel definition of *error*$(H, \mathcal{S})$ that significantly improves estimation quality.

Counting Independence Assumptions: The *nInd* error function (adapted from [4]) is simple and intuitive. We define the *error* of $\mathcal{S} = \mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_n}(P_n|Q_n)$, when each factor $\mathcal{S}el_{\mathcal{R}_i}(P_i|Q_i)$ is approximated using $SIT_{\mathcal{R}_i}(A_i|Q'_i)$ $(Q'_i \subseteq Q_i)$, as the total number of independence assumptions in the approximation. Formally,

$$\textit{nInd}(\{(\mathcal{S}_i, H_i)\}) = \sum_i |P_i| \cdot |Q_i - Q'_i|$$

where each term above represents the fact that P_i and $Q_i - Q'_i$ are assumed to be independent with respect to Q_i, and therefore the

number of independence assumptions is given by $|P_i| \cdot |Q_i - Q'_i|$. For instance, *nInd*$(\{\mathcal{S}el_{\mathcal{R}}(p|q_1,q_2), SIT_{\mathcal{R}}(p|q_1)\})$ is equal to 1 (i.e., one independence assumption). It is easy to see that *nInd* is monotonic and algebraic (according to Definition 3, we have that $e_i = |P_i| \cdot |Q_i - Q'_i|$ and E is the sum operator). Clearly, *nInd* is a syntactic definition that results in rough error approximations, but it can be computed very efficiently.

3.3 Approximating a Selectivity Factor

In general, the statistical information needed to estimate a given $\mathcal{S}el_{\mathcal{R}}(P|Q)$ consists of multiple SITs: although simple filter conditions can be approximated using a single SIT, join predicates in general require at least two SITs. For simplicity, we slightly modify the notation to represent SITs as follows. Consider query expression $q = \sigma_{p_1 \wedge \ldots \wedge p_k}(\mathcal{R}^\times)$. We use SIT$(a_1, \ldots, a_j|q)$ and SIT$_{\mathcal{R}}(a_1, \ldots, a_j|p_1, \ldots, p_k)$ interchangeably. That is, we enumerate the set of predicates in q over $\mathcal{R}^\times$, which agrees with the notation for selectivity values. Thus, $SIT_{\mathcal{R}}(a_1, \ldots, a_j|p_1, \ldots, p_k)$ expresses a histogram over attributes $\{a_1, \ldots, a_j\}$ built on the result of executing $\sigma_{p_1 \wedge \ldots \wedge p_k}(\mathcal{R}^\times)$. If there are no predicates p_i in q we write $SIT_{\mathcal{R}}(a_1, \ldots, a_j)$, which is a traditional base-table histogram. We now review the notion of *predicate independence* that we use to define the set of candidate SITs to consider for approximating a given selectivity value.

DEFINITION 4 (PREDICATE INDEPENDENCE). *Let P_1, P_2, and Q be sets of predicates. Then, P_1 and P_2 are* independent *with respect to Q if $\mathcal{S}el_{\mathcal{R}}(P_1, P_2|Q) = \mathcal{S}el_{\mathcal{R}_1}(P_1|Q) \cdot \mathcal{S}el_{\mathcal{R}_2}(P_2|Q)$, where $\mathcal{R}_1$=tables(P_1, Q) and $\mathcal{R}_2$=tables(P_2, Q).*

It is fairly easy to show that if P_1 and P_2 are independent with respect to Q, then $\mathcal{S}el_{\mathcal{R}}(P_1|P_2, Q) = \mathcal{S}el_{\mathcal{R}_1}(P_1|Q)$. We use independence between predicates as follows. Consider $\mathcal{S}el_{\mathcal{R}}(P|Q)$ and suppose there are no available SITs approximating $\mathcal{S}el_{\mathcal{R}}(P|Q)$, but there are SITs approximating $\mathcal{S}el_{\mathcal{R}}(P|Q')$ for some $Q' \subset Q$. Then, we implicitly *assume* independence between P and Q' with respect to $Q - Q'$ and use those SITs to approximate $\mathcal{S}el_{\mathcal{R}}(P|Q)$. Using this idea we now define the candidate set of SITs to approximate $\mathcal{S}el_{\mathcal{R}}(P|Q)$.

Filter Predicates

Let $\mathcal{S} = \mathcal{S}el_{\mathcal{R}}(P|Q)$, where P is a set of filter predicates, such as $\{R.a{=}5 \wedge S.b > 8\}$. The candidate SITs to approximate $\mathcal{S}$ are all $SIT_{\mathcal{R}}(A|Q')$ that satisfy:

1. *attr*$(P) \subseteq A$ (i.e., the SIT supports the predicates).
2. $Q' \subseteq Q$ (i.e., the SIT is consistent with the input query). In this case, we assume independence between P and Q-Q'. In a traditional optimizer, $Q' = \emptyset$, so P and Q are always assumed to be independent.
3. Q' is *maximal*, that is, there is no available $SIT_{\mathcal{R}}(A|Q'')$ such that $Q' \subset Q'' \subseteq Q$.

The set of candidate SITs can be defined in a more flexible way (e.g., accepting $SIT_{\mathcal{R}}(A|Q')$, where Q' *subsumes* Q [14]). We only consider the candidate SITs described above since they provide a good balance between the simplicity of the procedures to identify candidate SITs and the resulting approximation quality.

EXAMPLE 2. *Consider $\mathcal{S} = \mathcal{S}el_{\mathcal{R}}(R.a < 5|p_1, p_2)$ and the following SITs: $SIT_{\mathcal{R}}(R.a)$, $SIT_{\mathcal{R}}(R.a|p_1)$, $SIT_{\mathcal{R}}(R.a|p_2)$, and $SIT_{\mathcal{R}}(R.a|p_1, p_2, p_3)$. In this case, the set of candidate SITs for $\mathcal{S}$ include both $SIT_{\mathcal{R}}(R.a|p_1)$ and $SIT_{\mathcal{R}}(R.a|p_2)$. In contrast, $SIT_{\mathcal{R}}(R.a)$ does not qualify since its query expression is not maximal. Similarly, $SIT_{\mathcal{R}}(R.a|p_1, p_2, p_3)$ does not qualify since it contains an additional predicate p_3.* ∎

Filter and Join Predicates

Consider a selectivity factor $\mathcal{S}el_{\mathcal{R}}(P|Q)$ where P contains both filter and join predicates (e.g., $P{=}\{R.a < 5, R.x{=}S.y, S.b > 5\}$). We use the following observation about histograms. Let $H_1 = SIT_{\mathcal{R}}(x, X|Q)$ and $H_2 = SIT_{\mathcal{R}}(y, Y|Q)$ be SITs. A *histogram join* $H_1 \bowtie_{x=y} H_2$ returns not only the value $\mathcal{S}el_{\mathcal{R}}(x{=}y|Q)$ for the join, but also a new histogram $H_3 = SIT_{\mathcal{R}}(x, X, Y|x{=}y, Q)$. Therefore, we can use H_3 to estimate the remaining predicates involving attributes $x(=y)$, X, and Y.

EXAMPLE 3. *Consider $\mathcal{S}el_{\mathcal{R}}(R.a < 5, R.x{=}S.y|Q)$ and SITs $H_1 = SIT_{\mathcal{R}_1}(R.x, R.a|Q)$ and $H_2 = SIT_{\mathcal{R}_2}(S.y|Q)$. The join $H_1 \bowtie_{R.x=S.y} H_2$ returns a scalar value s_1 representing selectivity $\mathcal{S}el_{\mathcal{R}}(R.x = S.y|Q)$, and also $H_3{=}SIT_{\mathcal{R}}(R.a|R.x{=}S.y, Q)$. We then estimate s_2, the selectivity of filter predicate $(R.a < 5)$ using H_3, and obtain $\mathcal{S}el_{\mathcal{R}}(R.x{=}S.y, R.a < 5|Q) = s_1 \cdot s_2 = \mathcal{S}el_{\mathcal{R}}(R.a < 5|R.x{=}S.y, Q) \cdot \mathcal{S}el_{\mathcal{R}}(R.x{=}S.y|Q)$ (implicitly applying an atomic decomposition).* ∎

As the example shows, we can approximate $\mathcal{S}el_{\mathcal{R}}(P|Q)$ by getting SITs covering all attributes in P, joining such SITs accordingly, and finally estimating the remaining range predicates in P. In general, the set of candidate SITs to approximate $\mathcal{S}el_{\mathcal{R}}(P|Q)$ is *conceptually* obtained as follows:

1. We transform all join predicates in P to pairs of *wildcard selection predicates*. For the query in Example 3, predicate $R.x = S.y$ is replaced by $\{R.x =?, S.y =?\}$, obtaining the selectivity expression $\mathcal{S}el_{\mathcal{R}}(R.x =?, S.y =?, R.a < 5|Q)$. Wildcard selection predicates represent the fact that to estimate a join we need to obtain cardinality estimates for many different regions in the join column's domains, and the estimates for each column could in principle be provided by different histograms.

2. Let P' be the set of predicates obtained in the previous step. Because we replaced join predicates with filter predicates, the resulting selectivity expression becomes separable. We apply the separable decomposition property to $\mathcal{S}el_{\mathcal{R}}(P'|Q)$ and obtain $\mathcal{S}el_{\mathcal{R}_1}(P'_1|Q_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_k}(P'_k|Q_k)$, where no factor $\mathcal{S}el_{\mathcal{R}_i}(P'_i|Q_i)$ is separable. For our running example, we obtain $\mathcal{S}el_{\mathcal{R}_1}(R.x{=}?, R.a < 5|Q_1) \cdot \mathcal{S}el_{\mathcal{R}_2}(S.y{=}?|Q_2)$.

3. Now, each $\mathcal{S}el_{\mathcal{R}_i}(P'_i|Q_i)$ contains only filter predicates in P'_i, so we find each candidate set of SITs independently as described earlier. Then, to approximate the original selectivity value with the chosen SITs $\{H_i\}$, we first join all H_i on the selection wildcard predicate attributes, and then estimate the actual range predicates over the result, as illustrated in Example 3.

Obtaining the best candidates

Once we obtain the candidate set of SITs to approximate a selectivity factor $\mathcal{S}el_{\mathcal{R}}(P|Q)$, we simply select the alternative that is expected to result in the most accurate estimation for $\mathcal{S}el_{\mathcal{R}}(P|Q)$ according to the definition of *error* (see Section 3.2). In other words, we choose the alternative $\mathcal{H}$ that minimizes *error*$(\mathcal{H}, \mathcal{S}el_{\mathcal{R}}(P|Q))$. In the next section we use the concepts introduced so far and describe an algorithm to obtain the most accurate decomposition of a selectivity expression.

3.4 Algorithm *getSelectivity*

We now introduce *getSelectivity*($\mathcal{R}, P$), a dynamic programming algorithm that obtains the most accurate estimation of $\mathcal{S}el_{\mathcal{R}}(P)$ for a given *error* function. Our technique relies on the *error* function being monotonic and algebraic, and avoids considering decompositions with separable factors (see Sections 3.1 and 3.2).

Algorithm *getSelectivity* is shown in Figure 3.4. Lines 1-2 test whether the desired selectivity value was previously calculated, and if so returns it using a lookup in the memoization table. Otherwise, lines 4-7 handle the case in which $\mathcal{S}el_{\mathcal{R}}(P)$ is separable. Lines 4-5 obtain the standard decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ (Lemma 2) and recursively call *getSelectivity* for each factor $\mathcal{S}el_{\mathcal{R}_i}(P_i)$. Then, lines 6-7 combine the partial results. Otherwise (if $\mathcal{S}el_{\mathcal{R}}(P)$ is non-separable), lines 9-17 evaluate *all* atomic decompositions of $\mathcal{S}el_{\mathcal{R}}(P)$=$\mathcal{S}el_{\mathcal{R}}(P'|Q)\cdot\mathcal{S}el_{\mathcal{R}}(Q)$. For that purpose, line 11 recursively obtains the most accurate estimation (and the corresponding error) for $\mathcal{S}el_{\mathcal{R}}(Q)$ and line 12 locally obtains the best SIT H to approximate $\mathcal{S}el_{\mathcal{R}}(P'|Q)$ among the set of available SITs (see Section 3.3). If no SITs are available for approximating $\mathcal{S}el_{\mathcal{R}}(P'|Q)$, we set $\texttt{error}_{p|Q} = \infty$ and continue with the next atomic decomposition. Lines 13-15 keep track of the most accurate decomposition for $\mathcal{S}el_{\mathcal{R}}(P)$, and after exploring all atomic decompositions, lines 16-17 obtain the most accurate estimation for $\mathcal{S}el_{\mathcal{R}}(P)$. In all cases, before returning $\mathcal{S}el_{\mathcal{R}}(P)$ and its associated error in line 19, *getSelectivity* stores these values in the memoization table. As a byproduct of *getSelectivity*($\mathcal{R}, P$), we get the most accurate selectivity estimation for every sub-query $\sigma_{P'}(\mathcal{R}^{\times})$ with $P' \subseteq P$. In Section 4 we exploit these "free" selectivity estimates when integrating *getSelectivity* with existing optimizers.

THEOREM 1. *Algorithm getSelectivity*($\mathcal{R}, P$) *returns the most accurate approximation of* $\mathcal{S}el_{\mathcal{R}}(P)$ *for a given definition of error among all non-separable decompositions.*

The worst-case complexity of *getSelectivity*($\mathcal{R}, P$), with $|P|$=n, is $\mathcal{O}(3^n)$. In fact, the number of *different* calls of *getSelectivity* is at most 2^n, one for each subset of P. Due to memoization, only the first call for each subset of P actually produces some work. The running time of *getSelectivity* for k input predicates (not counting recursive calls) is $\mathcal{O}(k^2)$ in lines 4-7 and $\mathcal{O}(2^k)$ in lines 9-17. Therefore, the complexity of *getSelectivity* is $\mathcal{O}(\sum_{k=1}^{n}\binom{n}{k}\cdot 2^k)$, or $\mathcal{O}(3^n)$. We contrast the worst-case complexity of *getSelectivity*, $\mathcal{O}(3^n)$, with the lower bound of possible decompositions of Section 2, $\mathcal{O}((n+1)!)$. Since $(n+1)!/3^n$ is $\Omega(2^n)$, by using monotonic *error* functions we obtain an exponential decrease in the number of decompositions that are explored without missing the best one. If many subsets of P are separable, the running time is further reduced, since we solve strictly smaller problems independently.

We note that *getSelectivity*, as defined in Figure 3.4, iterates over every (different) non-separable decomposition to obtain the most accurate selectivity estimation of an input query. This is not strictly required and, in certain situations, the running time of *getSelectivity* can be reduced by further pruning the search space. While this is not the focus of this paper, we briefly comment one approach next. If the number of available SITs is small, those SITs can drive the search for the best decomposition instead of blindly trying, in lines 10-15, a large number of atomic decompositions that are known not to be successful. Specifically, instead of trying every decomposition $\mathcal{S}el_{\mathcal{R}}(P'|Q)\cdot\mathcal{S}el_{\mathcal{R}}(Q)$ in line 10, we can only explore those decompositions that could be approximated using some available SIT. For instance, suppose that the only SIT available is SIT($R.a|R \bowtie S$). In this case, to approximate the value $\mathcal{S}el_{\mathcal{R}}(R.a < 10, S.b > 5, R \bowtie S)$, line 10 should only explore decomposition $\mathcal{S}el_{\mathcal{R}}(R.a < 10|S.b > 5, R \bowtie S)\cdot\mathcal{S}el_{\mathcal{R}}(S.b > 5, R \bowtie S)$. Other atomic decompositions, such as $\mathcal{S}el_{\mathcal{R}}(S.b > 5|R.a < 10, R \bowtie S)\cdot\mathcal{S}el_{\mathcal{R}}(R.a < 10, R \bowtie S)$ can be safely discarded.

```
getSelectivity (R : tables, P : predicates over R^×)
returns (Sel_R(P), error_P) with minimum error_P.
01 if (Sel_R(P) was already calculated)
02    (Sel_R(P), error_P) = memo_table_lookup(P)
03 else if Sel_R(P) is separable
04    get the standard decomposition of Sel_R(P),
         Sel_R(P) = Sel_{R_1}(P_1) · ... · Sel_{R_n}(P_n)
05    (S_{P_i}, error_{P_i}) = getSelectivity(R_i, P_i), i = 1..n
06    S_P = S_{P_1} · ... · S_{P_n}
07    error_P = E_merge(error_{P_1}, ..., error_{P_n})
08 else   // Sel_R(P) is non-separable
09    error_P = ∞; bestH = NULL
10    for each P' ⊆ P, Q = P − P'
       // consider atomic decomposition Sel_R(P'|Q) · Sel_R(Q)
11       (S_Q, error_Q) = getSelectivity(R, Q)
12       (H, error_{P'|Q}) = best SIT and error for
                Sel_R(P'|Q)    // see Sections 3.3 and 3.5
13       if (E_merge(error_{P'|Q}, error_Q) ≤ error_P)
14          error_P=E_merge(error_{P'|Q}, error_Q)
15          bestH = H
16    S_{P'|Q}= estimation of Sel_R(P'|Q) using bestH
17    S_P = S_{P'|Q} · S_Q
18 memo_table_insert(P, S_P, error_P)
19 return (S_P, error_P)
```

Figure 3: Obtaining the most accurate selectivity estimation.

Line 12 in *getSelectivity* obtains the SITs H that minimize the value *error*($H, \mathcal{S}el_{\mathcal{R}}(p|Q)$) (see Section 3.3). In Section 3.2 we adapted a simple *error* function from [4]. We now introduce a novel formulation of *error* that results in better estimations.

3.5 *Diff*: An Improved Error Function

A critical subroutine in *getSelectivity* is *error*($\mathcal{H}, \mathcal{S}$), which models the estimated accuracy of approximating selectivity $\mathcal{S}$ using SITs $\mathcal{H}$. We identify two requirements for any *error* function:

Coarseness of available information: At first sight, it is tempting to reformulate *error* as a meta-estimation problem: to estimate the error between actual selectivity values and SIT-approximated selectivity values, we could maintain meta-statistics over the difference of such distributions. Thus, estimating *error*($\mathcal{H}, \mathcal{S}$) would be equivalent to approximate range queries over these meta-statistics. However, this approach is flawed, since if we do have such meta-statistics, we could combine them with the original SITs and obtain more accurate results in the first place. For instance, consider $\mathcal{S} = \mathcal{S}el_{\mathcal{R}}(R.a < 10|p_1, p_2)$ being approximated by $\mathcal{H} = SIT_{\mathcal{R}}(R.a|p_1)$. If we have available meta-statistics $\mathcal{M}$ to estimate values *error*($\mathcal{H}, \mathcal{S}el_{\mathcal{R}}(c_1 \le R.a \le c_2|p_1, p_2)$), we can combine $\mathcal{H}$ and $\mathcal{M}$ to obtain new SITs that directly approximate $\mathcal{S}el_{\mathcal{R}}(R.a < 10|p_1, p_2)$.

Efficiency: Evaluating *error*($\mathcal{H}, \mathcal{S}$) values must be efficient, since *error* is called repeatedly in the inner loop of *getSelectivity*. Very accurate but inefficient *error* functions are not useful, since the overall optimization time would increase and therefore exploiting SITs would become less attractive.

Therefore, we need efficient and coarse mechanisms to estimate *error* values. The *nInd* metric (see Section 3.2) satisfies these two properties. However, many alternatives often result in the same *nInd* value, and we need to break ties arbitrarily. This behavior is problematic when there are two or more available SITs to approximate a selectivity value, and while they result in the same "syn-

tactic" *nInd* score, the actual benefit of using each one of them is drastically different, as illustrated in the following example.

EXAMPLE 4. *Consider* $R \vec{\bowtie}_{R.s=S.s} (\sigma_{S.a<10} S) \vec{\bowtie}_{S.t=T.t} T$, *where both joins are defined between primary and foreign keys. Also consider the following factor of a decomposition for the query:* $\mathcal{S}_1$=$\mathcal{S}el_{\{R,S,T\}}(S.a < 10 | R \bowtie S, S \bowtie T)$. *Suppose that the only candidates to approximate* $\mathcal{S}_1$ *are* H_1=$SIT_{\{R,S\}}(S.a|R \bowtie S)$ *and* H_2=$SIT_{\{R,S\}}(S.a|S \bowtie T)$. *If we use nInd, error*$(\mathcal{S}_1, H_1) =$ *error*$(\mathcal{S}_1, H_2) = 1/2$, *so in general each alternative would be arbitrarily chosen. However,* H_1 *is a much better choice than* H_2. *In fact, since* $S \bowtie_{S.t=T.t} T$ *is a foreign-key join, the distribution of* $S.a$ *over the result of* $S \bowtie_{S.t=T.t} T$ *is exactly the same as the distribution of* $S.a$ *over base table* S. *Therefore,* $S \bowtie_{S.t=T.t} S$ *is actually independent of* $S.a < 10$, *and* H_2 *provides no benefit over the base histogram* $H(S.a)$. ∎

Inspired by the example above, we define an improved error function, *Diff*, as follows. Suppose first that we associate with each available SIT H=$SIT_{\mathcal{R}}(R.a|Q)$ a single value $0 \leq$ *diff*$_H \leq 1$ that measures the discrepancy between the distribution of $R.a$ on the base table and that of $R.a$ on the result of executing query expression Q. In particular, *diff*$_H = 0$ when the two distributions are the same, and *diff*$_H$ grows up to 1 when such distributions are very different (note that, in general, there are multiple possible distributions for which *diff*$_H = 1$, but only one for which *diff*$_H = 0$). Consider H=$SIT_{\mathcal{T}}(R.a|Q)$. If we denote $\mathcal{T}'$ to the result of evaluating Q over $\mathcal{T}^\times$ (i.e., $\mathcal{T}' = \sigma_Q(\mathcal{T}^\times)$), we define *diff* as follows[2]:

$$\textit{diff}_H = 1/2 \cdot \sum_{x \in dom(a)} \left| \frac{f(R,x)}{|R|} - \frac{f(\mathcal{T}',x)}{|\mathcal{T}'|} \right|$$

where $f(R,x)$ and $f(\mathcal{T}',x)$ are the frequencies of value x in R and $\mathcal{T}'$ respectively. The value *diff*$_H$ measures the deviation of frequencies between the base table distribution and the result of executing H's query expression. Values of *diff* are calculated just once and stored with each SIT, so there is no overhead at runtime. We can calculate *diff*$_{SIT_{\mathcal{R}}(a|Q)}$ when we create $SIT_{\mathcal{R}}(a|Q)$ by inspecting actual data tuples, but that might impose a certain overhead to the query processor to get values $f(R,a)$ if the tuples are not sorted by attribute a. Instead, we approximate *diff*$_H$ by manipulating both $SIT_{\mathcal{R}}(a|Q)$ and the corresponding base-table histogram on column a. This procedure is similar to techniques that approximate joins using histograms, but we omit the details due to space constraints.

Using *diff* values, the *Diff* error function provides a less syntactic notion of independence. In particular, the overall *error* value for a decomposition $\mathcal{S} = \mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_n}(P_n|Q_n)$ when approximated using $H_1, \ldots, H_n$ is:

$$\textit{Diff}(\{(\mathcal{S}_i, H_i)\}) = \sum_i |P_i| \cdot (1 - \textit{diff}_{H_i})$$

Intuitively, the values $(1 -$ *diff*$_{H_i})$ above represent the "semantic" degree of independence when approximating $\mathcal{S}_i$ with H_i, and replace the "syntactic" value $|Q_i - Q_i'|$ of *nInd* (see Section 3.2). In Example 4, *diff*$_{H_2} = 0$, and H_2 effectively contributes the same as a base-table histogram $H(S.a)$, so in that case the error function is 1 (the maximum possible value). In contrast, for H_1=$SIT_{\{R,\,S\}}(S.a|R \bowtie S)$, the more different the distributions of $S.a$ on S and on the result of executing $R \bowtie S$, the more likely that H_1 encodes dependencies between $S.a$ and $\{R \bowtie S, S \bowtie T\}$, and the lower the overall error value.

[2]A similar metric, μ_{count}, is proposed in [13] to compare two histogram distributions.

Of course, *Diff* is just a heuristic ranking function and has some natural limitations. For instance, it uses a single number (*diff*$_H$) to summarize the amount of divergence between two distributions. However, as we will see in Section 5, *Diff* is much more robust and accurate than *nInd* with almost no additional overhead.

4. INTEGRATION WITH AN OPTIMIZER

In this section we show how *getSelectivity* can be integrated with rule-based optimizers. For $q = \sigma_{p_1 \wedge \ldots \wedge p_k}(\mathcal{R}^\times)$, *getSelectivity* returns the most accurate selectivity estimation for both q and all its sub-queries (i.e., $\mathcal{S}el_{\mathcal{R}}(P)$ for all $P \subseteq \{p_1, \ldots, p_k\}$). A simple approach to incorporate *getSelectivity* into an existing rule-based optimizer is to execute *getSelectivity before* optimization starts, and then use the resulting memoization table to answer selectivity requests over arbitrary sub-queries. Instead, we propose to interleave the execution of *getSelectivity* with the optimizer's own search strategy. This way, *getSelectivity* can be integrated into current optimizers with very small changes. In Section 4.1 we describe Cascades, a framework used in current optimizers. Then, in Section 4.2 we show how we can couple *getSelectivity* with the search strategy of a Cascades-based optimizer.

4.1 Cascades-based Optimization

Cascades is one state-of-the-art rule-based optimization framework used in current optimizers such as Tandem's NonStop SQL and Microsoft SQL Server. During the optimization of an input query, a Cascades-based optimizer keeps track of many alternative sub-plans that could be used to evaluate the query. Sub-plans are grouped into equivalence classes, and each equivalence class is stored as a separate node in a memoization table. Thus, each node in the memo contains a list of logically equivalent alternatives explored so far. Each entry in a memo node has the form

$$\big[op,\ \{parm_1, \ldots, parm_k\}, \{input_1, \ldots, input_n\}\big]$$

where *op* is a logical operator, such as `join`, $parm_i$ are parameters for the operator (such as the range for a `filter` operator, or the columns for a `join operator`), and $input_j$ are pointers to other memo nodes (i.e., classes of equivalent sub-queries) that represent the input values to the operator.

EXAMPLE 5. *The memo node at the top of Figure 4 groups together all the query plans explored so far that are equivalent to* $(\sigma_{R.a<10}(R)) \bowtie_{R.x=S.y} (\sigma_{S.b>20}(S))$. *The first entry in this top node,* $\big[$`SELECT`$, \{R.a < 10\}, \{R \bowtie_{R.x=S.y} (\sigma_{S.b>20}(S))\}\big]$, *corresponds to a filter operator, with parameter* $R.a < 10$, *applied to the node that groups all equivalent alternatives to sub-expression* $R \bowtie_{R.x=S.y} (\sigma_{S.b>20}(S))$. *Similarly, the second entry in the top node corresponds to a join operator applied to two other nodes.* ∎

During optimization, each node in the memo is populated by applying *transformation rules* to the set of explored alternatives. Rules consist of antecedent and consequent patterns, and optional applicability conditions. For instance, the first entry at the top node of Figure 4 could have been obtained from the second entry by applying the transformation rule below, which pulls out selections above join predicates:

$$[T_1] \bowtie (\sigma_P[T_2]) \Rightarrow \sigma_P([T_1] \bowtie [T_2])$$

where T_1 and T_2 are placeholders for arbitrary sub-queries.

4.2 Integrating *getSelectivity* with Cascades

We now discuss how algorithm *getSelectivity* can be integrated with a Cascades-based optimizer. Specifically, we couple the execution of *getSelectivity* with the optimizer's own search strategy. As

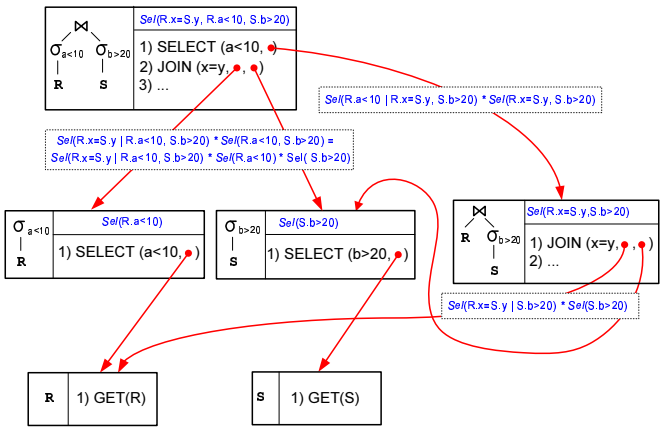


Figure 4: Intermediate memo in a Cascades-based optimizer.

we will see, the pruning is guided by the optimizer's own heuristics, and therefore might prevent *getSelectivity* from finding the most accurate estimation for some selectivity values. However, the overhead imposed to an existing optimizer is very small and, as we will see, the overall increase in quality can be substantial.

Consider an input SPJ query $q = \sigma_{p_1 \wedge \ldots \wedge p_k}(\mathcal{R}^{\times})$. As explained in Section 4.1, each node in the memoization table of the optimizer groups alternative representations of a sub-query of q. Therefore, we can estimate the selectivity of the sub-query represented by each node N in the memo, or $\mathcal{S}el_{\mathcal{R}}(P)$ for $P \subseteq \{p_1, \ldots, p_k\}$. More importantly, each *entry* in N is associated with a *decomposition* of the sub-query represented by N. This is illustrated below.

EXAMPLE 5. **(cont.)** *Consider again the node at the top of Figure 4, which groups all equivalent alternatives for evaluating* $(\sigma_{R.a<10}(R)) \bowtie_{R.x=S.y} (\sigma_{S.b>20}(S))$. *The second entry in such node (the* `join` *operator) can be associated with decomposition* $\mathcal{S}el_{\{R,S\}}(R.x = S.y | R.a < 10, S.b > 20) \cdot \mathcal{S}el_{\{R,S\}}(R.a < 10, S.b > 20)$. *The first factor of this decomposition is approximated using available SITs as explained in Section 3. The second factor is separable as* $\mathcal{S}el_{\{R\}}(R.a < 10) \cdot \mathcal{S}el_{\{S\}}(S.b > 20)$. *We obtain the estimated selectivity of each factor of the separable decomposition above by simply examining the corresponding memo nodes (the input of the* `join` *entry we are processing). Finally, we multiply such estimations and the first factor of the atomic decomposition* $\mathcal{S}el_{\{R,S\}}(R.x{=}S.y | R.a < 10, S.b > 20)$ *to obtain a new estimation for* $\mathcal{S}el_{\{R,S\}}(R.x{=}S.y, R.a < 10, S.b > 20)$. ∎

As hinted in the previous example, each entry $\mathcal{E}$ in a memo node N divides the set of predicates P that are represented by N into two groups: (i) the parameters of $\mathcal{E}$, denoted $p_{\mathcal{E}}$, and (ii) the predicates in the set that are input to $\mathcal{E}$, denoted $Q_{\mathcal{E}}$=P–$p_{\mathcal{E}}$. We then associate the decomposition $\mathcal{S}el_{\mathcal{R}}(P) = \mathcal{S}el_{\mathcal{R}}(p_{\mathcal{E}}|Q_{\mathcal{E}}) \cdot \mathcal{S}el_{\mathcal{R}}(Q_{\mathcal{E}})$ with entry $\mathcal{E}$. We note that to obtain the best SITs to approximate $\mathcal{S}el_{\mathcal{R}}(p_{\mathcal{E}}|Q_{\mathcal{E}})$ we can reuse the view matching sub-routines in the optimizer and exploit the *property derivation* framework in Cascades, but we omit those details in this paper. It is fairly easy to show that for each operator in $\mathcal{E}$, $\mathcal{S}el_{\mathcal{R}}(Q_{\mathcal{E}})$ is separable into $\mathcal{S}el_{\mathcal{R}_1}(Q^1_{\mathcal{E}}) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_k}(Q^k_{\mathcal{E}})$, where each $\mathcal{S}el_{\mathcal{R}_i}(Q^i_{\mathcal{E}})$ is associated with the i-th input of $\mathcal{E}$.

In summary, we restrict the set of decompositions in line 10 of *getSelectivity* to those induced by the optimization search strategy. Each time we apply a transformation rule that results in a new entry $\mathcal{E}$ in the node associated with $\mathcal{S}el_{\mathcal{R}}(P)$, we obtain the decomposition $\mathcal{S} = \mathcal{S}el_{\mathcal{R}}(p_{\mathcal{E}}|Q_{\mathcal{E}}) \cdot \mathcal{S}el_{\mathcal{R}}(Q_{\mathcal{E}})$, and keep the decomposition with the best estimated accuracy. We discuss the overhead imposed to a traditional optimizer experimentally in the following section.

5. EXPERIMENTS

In this section, we experimentally study *getSelectivity* using both *nInd* and *Diff* as the underlying error functions, and compare it against the approach of [4]. As we will see, using conditional selectivity values results in more accurate estimations than when solely relying on materialized view matching to exploit SITs. Also, when using *Diff*, the *getSelectivity* approximations are close to being optimal. For our experiments we use the following setting:

Data Sets: We generate a synthetic database with a snowflake schema, consisting of 8 tables with 1K to 1M tuples and 4 to 8 attributes. Attribute values are generated with different degrees of skew and correlation. Additionally, some foreign-key joins do not satisfy referential integrity. In such situations, for a foreign-key join $R \vec{\bowtie} S$, we chose a certain percentage of tuples in R (between 5% and 20%) and replace the join attribute in those tuples with NULL values. The choice of the dangling tuples is either random or correlated with attribute values.

Workloads: Each workload consists of 100 randomly generated SPJ queries, with parameters J (number of join predicates) and F (number of filter predicates). In our experiments, we set F to three[3] and vary J from 3 to 7. For each query, we choose filter predicates such that the selectivity of each one is around 0.05. If the query result is empty, we progressively stretch the filter ranges until at least one tuple is present in the result. We also used predicates with selectivities around 0.5 and obtained similar trends, but we believe that those queries are less frequent in real applications, so we omit those results.

Available SITs: We experiment with different pools of available SITs. Each SIT is a unidimensional *maxDiff* histogram [22] with at most 200 buckets. Each set of SITs J_i contains all histograms of the form $SIT_{\mathcal{R}}(a|Q)$, where Q consists of *at most* i join predicates, and both Q and a are syntactically present in some query in the workload (J_0 contains all and only base-table histograms). The number of available SITs ranged from 82 (for J_1) to 680 (for J_7).

Techniques Compared: We implemented the technique of [4], which we refer to as *GVM* (for Greedy View Matching), and several variations of the algorithm *getSelectivity*. In particular, we implemented *GS-nInd*, the variation that counts the number of independence assumptions, and *GS-Diff*, the variation that uses additional information about the data distribution. We also implemented *GS-Opt*, which uses the actual difference between the true selectivity $\mathcal{S}$ and the approximation using $\mathcal{H}$ to define $error(\mathcal{H}, \mathcal{S})$: this definition of *error* is the best possible one, but is only of theoretical interest since it cannot be implemented efficiently (it inspects the actual data). Finally, we implemented *noSit*, which mimics current optimizers and exploits base-table histograms only.

Metrics: We compare the accuracy of the different techniques as follows. For each query q in the workload, we first estimate the cardinality of each sub-query of q using the different techniques. Then, we evaluate each sub-query to obtain its actual cardinality value, and finally obtain the average estimation error over all sub-series of q. We average this individual result over each query in the workload. A comprehensive study on how plans are affected by the estimation techniques proposed in this paper is part of future work.

5.1 Comparison with Previous Approaches

We first compare *GVM* against *GS-nInd*, so that the error function in *getSelectivity* agrees with that of the greedy procedure in *GVM*. Any difference in accuracy is then due to *getSelectivity* ex-

[3]We obtained similar results when using more filter predicates.

ploring the full search space, and not caused by an improved error function such as *Diff*. Figure 5 shows a two-dimensional graph, where each point represents one query in a workload consisting of 3- to 7-way join queries. The x- and y-axes represent the absolute cardinality error for each query using *GVM* and *getSelectivity*, respectively. As shown in the figure, all points lie under the line $x = y$, which means that *GS-nInd* results in consistently better cardinality estimates than *GVM*. The reason is that the search space in *GVM* is a strict subset of the space of decompositions explored by *GS-nInd*. In addition, *GVM* uses a greedy technique to iteratively select SITs, which further reduces the set of decompositions explored. Although *GS-nInd* is based on the same metric to rank candidate SITs, it can result in absolute errors that are as much as 80% lower than those of the *GVM* technique.

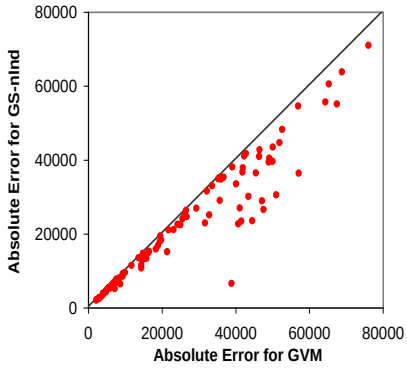


Figure 5: Accuracy of *GS-nInd* vs. *GVM*.

To compare the efficiency of both techniques, we proceed as follows. Both *GVM* and *getSelectivity* share the same view matching algorithm in their inner loops to select SITs (*GVM* during the greedy procedure, and *getSelectivity* as a subroutine in line 12). We then chose to compare the average number of calls to the view matching routine[4], shown in Figure 6 for both *getSelectivity* and *GVM*, and for different workloads. The dynamic programming algorithm used by *getSelectivity* results in drastically fewer calls, despite searching the whole space of decompositions. Although *GVM* is more efficient than *getSelectivity* for a single invocation, it does not exploit commonalities between different sub-plans, and thus results in even 5 times as many view matching calls as *getSelectivity*.

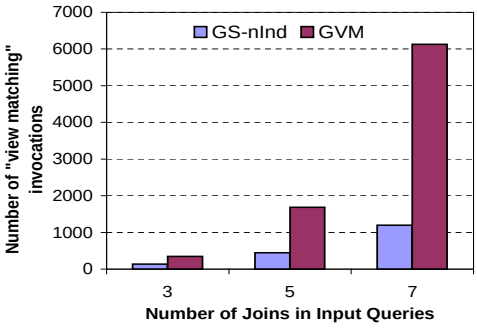


Figure 6: Efficiency of *GS-nInd* vs. *GVM*.

To summarize, in this section we experimentally established the superiority of *GS-nInd* over *GVM*, both in terms of accuracy and efficiency, for SPJ queries. We now take a closer look at *getSelectivity* and compare *GS-nInd* and *GS-Diff*.

[4]We also compared execution times and the trends were similar to those presented here.

5.2 Accuracy of *getSelectivity*

Figure 7 shows the average absolute error for workloads of 3-, 5-, and 7-way join queries. In all cases, the absolute error is reduced drastically when all SITs with join query expressions (J_7) are available (e.g., the average absolute error is reduced from 62,466 to 1,679 in Figure 7(a)). In particular, *GS-Diff* is very close to the optimal strategy *GS-Opt*, and results in considerably more accurate estimations than *GS-nInd*. For *GS-Diff*, the largest reduction in error occurs for J_1 and J_2. In our experiments, SITs with 2- and 3-way join query expressions are responsible for most of the accuracy gains.

5.3 Efficiency of *getSelectivity*

Figure 8 shows the average execution time of *GS-Diff* for different input workloads (results for *GS-nInd* are very similar). We partition the execution time of *getSelectivity* into two components. The *decomposition analysis* is the time spent to process the different decompositions and choose the best candidate SITs. The *histogram manipulation* corresponds to line 16 of *getSelectivity* and measures the actual estimation of selectivity values using the selected SITs. The reason for this division is that different techniques choose different histograms to estimate the same selectivity values. Different histograms take different amounts of time to estimate the same predicates, so we report these components separately.

Figure 8 shows that, in general, *getSelectivity* results in a small overhead over *noSit*, under 6 milliseconds in all scenarios (under 4 milliseconds if we also consider the histogram manipulation component). We note that the overhead of *getSelectivity* is proportional to the number of candidates considered in the algorithm, and scales gracefully with the number of available SITs. For instance, in Figure 8, *getSelectivity* executes in around 6 msecs. for J_1 (with 82 SITs), and in around 9 msecs. for J_7 (with 680 SITs).

In conclusion, we expect that the overhead of *getSelectivity* in a real optimizer will be very small, since Figure 8 accounts for just a portion of the total optimization time. Other components, such as sophisticated rule-based engines, also contribute to the overall optimization time.

6. RELATED WORK

Virtually all optimization frameworks (e.g., [15, 17, 24]), rely on statistics over base tables in the database to choose the most efficient execution plan in a cost-based manner. There is a large body of work that studies representation of statistics on a given column [18, 19, 21, 22] or combination of columns [1, 5, 16, 20, 23]. In this paper we rely on existing histogram techniques and focus on approximating attribute distributions over query expressions.

The idea of building statistics over non-base tables first appears (implicitly) in [2]. This reference introduces *join synopses*, which are pre-computed samples of a small set of distinguished joins. Joins must be defined between foreign and primary keys, and therefore a single sample for each table is enough to provide approximate answers for a large number of queries. The idea is to conceptually materialize the extended table obtained by applying all foreign-key joins, and then take a uniform sample over this result. Reference [11] extends this approach by introducing *icicles*, a new class of samples that tune themselves to a dynamic workload. Intuitively, the probability of a tuple being present in an icicle is proportional to its importance for answering queries in the workload.

References [4, 26] introduced the concept of statistics on query expressions (or views), and showed how to incorporate them into existing query optimizers. Specifically, the idea in [4] is to transform each input query sub-plan into an equivalent one that exploits

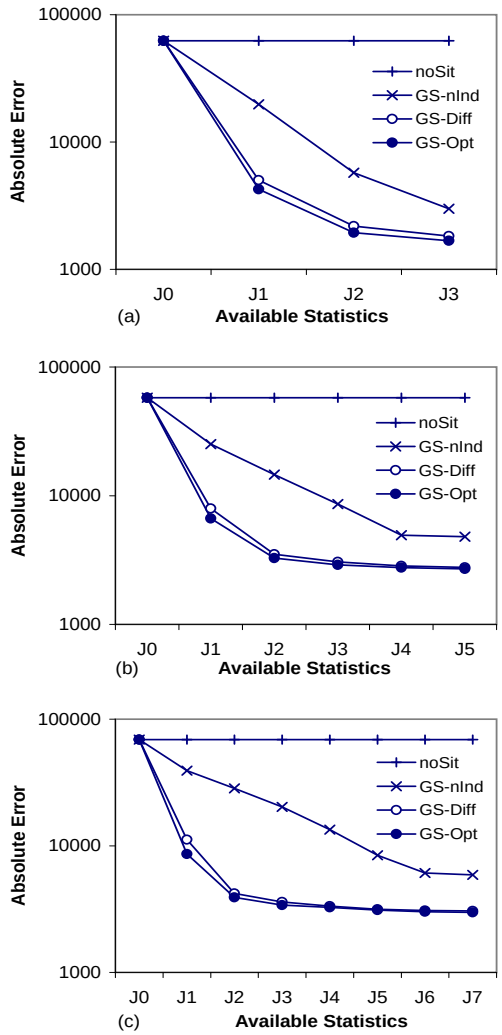


Figure 7: Average absolute error for (a) 3-, (b) 5-, and (c) 7-way join queries.

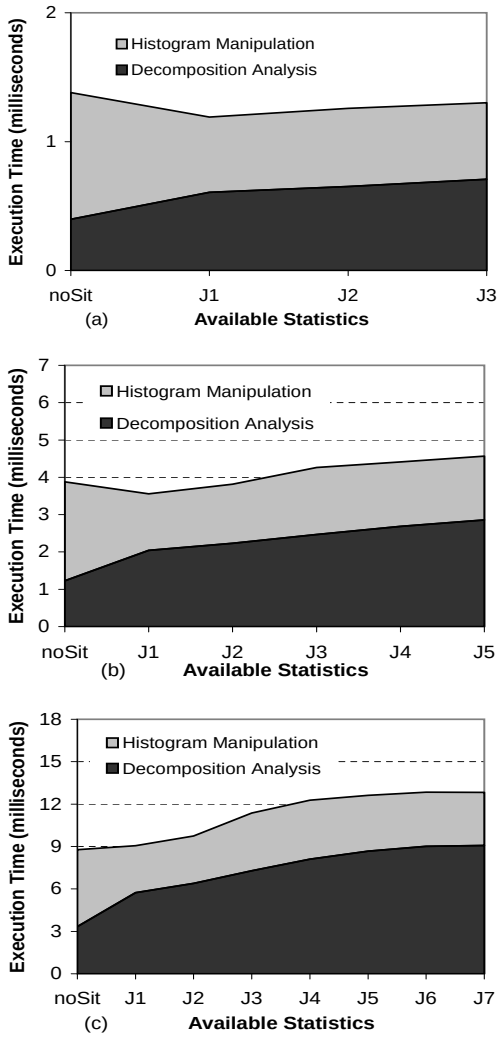


Figure 8: Execution time for (a) 3-, (b) 5-, and (c) 7-way join queries.

SITs, leveraging materialized view matching techniques [7, 14]. The transformation step is based on a greedy procedure that selects which SITs result in the transformed plan using the minimal number of independence assumptions. In this work we critically analyze the main drawbacks of the techniques in [4, 26]. We propose a novel framework to reason with SITs and an efficient algorithm that returns the optimal decomposition of a selectivity value.

Similar to previous work in self-tuning histograms [1, 5], reference [25] presents an online algorithm that repairs incorrect statistics and cardinality estimates of a query execution plan. By monitoring previously executed queries, [1, 5, 25] compute adjustments to base-table statistics that might be used during future query optimizations. The key difference with our approach is that [25] maintains a single adjusted histogram per attribute and still relies on the independence assumption during cardinality estimation. However, the adjustments are done in such a way that the cardinality of the processed query is correctly calculated despite assuming independence. Instead, we rely on different statistics for the same attribute depending on its particular context in the corresponding query plan.

References [8, 12] use conditional probability concepts to model multidimensional distributions. Reference [12] shows how to use probabilistic relational models to approximate the joint distribution of multiple attributes in one table, or in different tables that are combined using foreign-key joins. It uses the concept of conditional independence between attributes to decompose the representation of a join distribution into factors that capture the actual independencies that hold in the data domain, therefore obtaining a compact representation of the actual distribution. This work mostly focuses on point queries and assumes that each attribute has a small discrete domain. With a similar motivation, reference [8] uses statistical interaction models to explicitly identify and exploit the statistical characteristics of the underlying data. The rationale is that real tables are characterized by complex correlation patterns, where a certain subset of the attributes can be (unconditionally) independent of another attribute subset, or, alternatively, can be (conditionally) independent of given a third subset of attributes. The idea in [8] is to break the statistics (e.g., multidimensional histograms) into (i) an interaction model that accurately captures significant correlation and independence patterns in data, and (ii) a collection of lower-dimensional histograms that, based on the model, can provide accurate approximations of the overall joint data distribution. We believe that the ideas introduced in both [8] and [12] can be fairly easily incorporated into our framework of Section 3. If we can infer from the data distribution that

some predicates are conditionally independent of others, we can apply *semantic* versions of the "separable decomposition" property to further reduce the space of decompositions. As a simple example, consider $Sel_{\mathcal{R}}(p_1, p_2, p_3)$. If we can infer that p_1 is conditionally independent of p_2 given p_3 using the techniques described above, we can decompose the given selectivity value as $Sel_{\mathcal{R}}(p_1, p_2, p_3) = Sel_{\mathcal{R}}(p_1|p_2) \cdot Sel_{\mathcal{R}}(p_2, p_3)$ *without* relying on any assumption, such as independence.

7. SUMMARY

In this paper we introduced the novel framework of conditional selectivity to reason with selectivity values. This framework allows us to identify the space of decompositions to approximate selectivity values for a given set of available SITs with high efficiency and accuracy. We designed a dynamic programming algorithm, *getSelectivity*, that returns the most accurate selectivity estimation for an input query. *getSelectivity* can be integrated with existing optimizers by coupling its execution with the optimizer's own search strategy. Our preliminary experiments show that *getSelectivity* results in much more accurate estimations than previous approaches (both the ones that consider SITs or rely on base-table statistics only), and that the expected overhead of *getSelectivity* is small enough to increase overall performance of current optimizers.

Acknowledgments

We thank Luis Gravano and Cesar Galindo Legaria for providing valuable feedback on earlier versions of this paper.

8. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1999.

[2] S. Acharya et al. Join synopses for approximate query answering. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1999.

[3] N. Bruno. Statistics on query expressions in relational database management systems. Ph.D. thesis, Columbia University, 2003.

[4] N. Bruno and S. Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2002.

[5] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2001.

[6] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings of the International Conference on Data Engineering*, 2002.

[7] S. Chaudhuri et al. Optimizing queries with materialized views. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 1995.

[8] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2001.

[9] R. Fagin. Fuzzy queries in multimedia database systems. In *Proceedings of the Seventeenth ACM Symposium on Principles of Database Systems*, June 1998.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. of the Twentieth ACM Symposium on Principles of Database Systems*, 2001.

[11] V. Ganti, M.-L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2000.

[12] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2001.

[13] P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB)*, 1997.

[14] J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: A practical, scalable solution. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2001.

[15] G. Graefe. The Cascades framework for query optimization. *Data Engineering Bulletin*, 18(3), 1995.

[16] D. Gunopulos et al. Approximating multi-dimensional aggregate range queries over real attributes. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2000.

[17] L. M. Haas et al. Extensible query processing in Starburst. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1989.

[18] H. V. Jagadish et al. Optimal histograms with quality guarantees. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, 1998.

[19] A. C. Konig and G. Weikmun. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In *Proceedings of the 25th International Conference on Very Large Databases (VLDB)*, 1999.

[20] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1988.

[21] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1984.

[22] V. Poosala et al. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1996.

[23] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB)*, 1997.

[24] P. G. Selinger et al. Access path selection in a relational database management system. In *Proceedings of the ACM International Conference on Management of Data*, 1979.

[25] M. Stillger et al. LEO - DB2's learning optimizer. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB)*, 2001.

[26] F. Waas, C. Galindo-Legaria, M.-C. Wu, and M. Joshi. Statistics on views. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB)*, 2003.

APPENDIX

Proofs

PROPERTY 1 (ATOMIC DECOMPOSITION). *Given a set of tables $\mathcal{R}$ and sets of predicates P and Q:*

$$\mathcal{S}el_{\mathcal{R}}(P,Q) = \mathcal{S}el_{\mathcal{R}}(P|Q) \cdot \mathcal{S}el_{\mathcal{R}}(Q)$$

Proof: Using the definition of conditional selectivity, the above equality is expressed as:

$$\frac{|\sigma_{P\wedge Q}(\mathcal{R}^{\times})|}{|\mathcal{R}^{\times}|} = \frac{|\sigma_P(\sigma_Q(\mathcal{R}^{\times}))|}{|\sigma_Q(\mathcal{R}^{\times})|} \cdot \frac{|\sigma_Q(\mathcal{R}^{\times})|}{|\mathcal{R}^{\times}|}$$

or, equivalently,

$$|\sigma_{P\wedge Q}(\mathcal{R}^{\times})| = |\sigma_P(\sigma_Q(\mathcal{R}^{\times}))|$$

which always holds in relational algebra. ∎

LEMMA 1. *The number of possible decompositions of $\mathcal{S}el_{\mathcal{R}}(p_1,\ldots,p_n)$, denoted $T(n)$, is bounded by*

$$0.5 \cdot (n+1)! \leq T(n) \leq 1.5^n \cdot n!, \qquad \textit{for all } n \geq 1$$

Proof: The number of decompositions of $\mathcal{S}el_{\mathcal{R}}(P)$, where $|P| = n$, is given by the following equation:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{i=1}^{n} \binom{n}{i} \cdot T(n-i) & \text{otherwise} \end{cases}$$

In fact, for each $1 \leq i \leq n$, we can first decompose $\mathcal{S}el_{\mathcal{R}}(P)$ into $\mathcal{S}el_{\mathcal{R}}(P_1|P_2) \cdot \mathcal{S}el_{\mathcal{R}}(P_2)$, where $|P_1| = i$ and $|P_2| = n - i$. Then, we recursively obtain all decompositions for $\mathcal{S}el_{\mathcal{R}}(P_2)$. After some manipulation, it follows that

$$T(n+1) = \sum_{i=0}^{n} \binom{n+1}{i} \cdot T(i) =$$

$$(n+1) \cdot T(n) + \sum_{i=0}^{n-1} \frac{n+1}{n+1-i} \cdot \binom{n}{i} \cdot T(i)$$

The fraction in the summation above satisfy $1 \leq \frac{n+1}{n+1-i} \leq \frac{n+1}{2}$ (for $i = 0$ and $i = n-1$, respectively). From the first inequality, it follows that $T(n+1) \geq (n+1) \cdot T(n) + T(n) = (n+2) \cdot T(n)$. By solving this simpler recurrence, we conclude that $T(n+1) \geq 0.5 \cdot (n+2)!$, as desired. The remaining bound follows from an analogous analysis of the second inequality: $T(n+1) \leq 1.5^{n+1} \cdot (n+1)!$. ∎

PROPERTY 2 (SEPARABLE DECOMPOSITION). *Suppose that $\{P_1, P_2\}$ and $\{Q_1, Q_2\}$ are partitions of P and Q, respectively, and X_1=$P_1 \cup Q_1$ and $X_2 = P_2 \cup Q_2$ separate $\mathcal{S}el_{\mathcal{R}}(P|Q)$. Let $\mathcal{R}_1$=tables(X_1) and $\mathcal{R}_2$=tables(X_2). Then, $\mathcal{S}el_{\mathcal{R}}(P|Q) = \mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1) \cdot \mathcal{S}el_{\mathcal{R}_2}(P_2|Q_2)$.*

Proof: Let $\mathcal{T} = \mathcal{R} - (\mathcal{R}_1 \cup \mathcal{R}_2)$. We have that:

$$\mathcal{S}el_{\mathcal{R}}(P|Q) = \mathcal{S}el_{\mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{T}}(P_1, P_2|Q_1, Q_2) =_{\text{definition of } \textit{Sel}}$$

$$\frac{|\sigma_{P_1\wedge P_2}(\sigma_{Q_1\wedge Q_2}(\mathcal{R}_1^{\times} \times \mathcal{R}_2^{\times} \times \mathcal{T}^{\times}))|}{|\sigma_{Q_1\wedge Q_2}(\mathcal{R}_1^{\times} \times \mathcal{R}_2^{\times} \times \mathcal{T}^{\times})|} =_{\text{relational algebra}}$$

$$\frac{|\sigma_{P_1\wedge Q_1}(\mathcal{R}_1^{\times})| \cdot |\sigma_{P_2\wedge Q_2}(\mathcal{R}_2^{\times})| \cdot |\mathcal{T}^{\times}|}{|\sigma_{Q_1}(\mathcal{R}_1^{\times})| \cdot |\sigma_{Q_2}(\mathcal{R}_2^{\times})| \cdot |\mathcal{T}^{\times}|} =_{\text{definition of } \textit{Sel}}$$

$$\mathcal{S}el_{\mathcal{R}_1}(P_1|Q_1) \cdot \mathcal{S}el_{\mathcal{R}_2}(P_2|Q_2)$$

∎

LEMMA 2. *There is a unique decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$ as $\mathcal{S}el_{\mathcal{R}_1}(P_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_n}(P_n)$, where each $\mathcal{S}el_{\mathcal{R}_i}(P_i)$ is not separable. We call it the* standard decomposition of $\mathcal{S}el_{\mathcal{R}}(P)$.

Proof: Suppose that there are two standard decompositions of $\mathcal{S}$, namely $\mathcal{S}_1 = \mathcal{S}el_{\mathcal{R}_1}(P_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{R}_m}(P_m)$ and $\mathcal{S}_2 = \mathcal{S}el_{\mathcal{S}_1}(Q_1) \cdot \ldots \cdot \mathcal{S}el_{\mathcal{S}_n}(Q_n)$ of $\mathcal{S}el_{\mathcal{R}}(P)$. Since $\mathcal{S}_1$ and $\mathcal{S}_2$ are different and $\cup_i P_i = \cup_j Q_j = P$, there must be a pair (P_i, Q_j) such that $P_i \neq Q_j$ and $P_i \cap Q_j \neq \emptyset$. Let $X_{i,j} = P_i \cap Q_j$. Then, $X_{i,j}$ and $P_i - X_{i,j}$ separate $\mathcal{S}el_{\mathcal{R}_i}(P_i)$. To prove that, let $T_{i,j} = \textit{tables}(X_{i,j})$. It is fairly easy to show that $T_{i,j} \subseteq \mathcal{R}_i$ and $T_{i,j} \subseteq \mathcal{S}_j$. Also, by definition of $\mathcal{S}_2$, Q_j and $P - Q_j$ separate $\mathcal{S}el_{\mathcal{R}}(P)$, so no single predicate from P references attributes in $\mathcal{S}_j$ and $\mathcal{R} - \mathcal{S}_j$ simultaneously. Therefore, no predicate in P references attributes in $T_{i,j} \subseteq \mathcal{S}_j$ and $(\mathcal{R}_i - T_{i,j}) \subseteq (\mathcal{R} - \mathcal{S}_j)$, which essentially means that $X_{i,j}$ and $P_i - X_{i,j}$ separate $\mathcal{S}el_{\mathcal{R}_i}(P_i)$. Analogously, $T_{i,j}$ and $Q_j - T_{i,j}$ separate $\mathcal{S}el_{\mathcal{S}_j}(Q_j)$. Therefore, neither decomposition above is standard. ∎

THEOREM 1. *Algorithm getSelectivity$(\mathcal{R}, P)$ returns the most accurate approximation of $\mathcal{S}el_{\mathcal{R}}(P)$ for a given definition of error among all non-separable decompositions.*

Proof: [Sketch] We prove the theorem by induction on $|P|$. The base case ($|P| = 1$), is trivially verified. For the general case, if $\mathcal{S} = \mathcal{S}el_{\mathcal{R}}(P)$ is non-separable, line 10 exhaustively explores all atomic decompositions of $\mathcal{S}$. Using the inductive hypothesis and the principle of optimality of *error* we prove the inductive step for this case. Otherwise, if $\mathcal{S}$ is separable, we show that we do not miss any non-separable decomposition of $\mathcal{S}$ by processing instead its standard decomposition (lines 4-7). Consider $\mathcal{S} = \mathcal{S}el_{\mathcal{R}}(P, Q)$, where P and Q separate $\mathcal{S}$, and let $\mathcal{S}'$=$\mathcal{S}el_{\mathcal{R}}(P_1, Q_1|P_2, Q_2) \cdot \mathcal{S}el_{\mathcal{R}}(P_2, Q_2)$ be an arbitrary atomic decomposition of $\mathcal{S}$. Both factors of $\mathcal{S}'$ are separable, so applying the separable decomposition property it follows that $\mathcal{S}' = \mathcal{S}el_{\mathcal{R}}(P_1|P_2) \cdot \mathcal{S}el_{\mathcal{R}}(Q_1|Q_2) \cdot \mathcal{S}el_{\mathcal{R}}(P_2) \cdot \mathcal{S}el_{\mathcal{R}}(Q_2)$. We then know that $\mathcal{S}'$ is explored from $\mathcal{S}el_{\mathcal{R}}(P) \cdot \mathcal{S}el_{\mathcal{R}}(Q)$, the standard decomposition of $\mathcal{S}$. Therefore, we do not miss any non-separable decomposition and again, by using the inductive hypothesis and the monotonicity of *error*, we prove the theorem. ∎