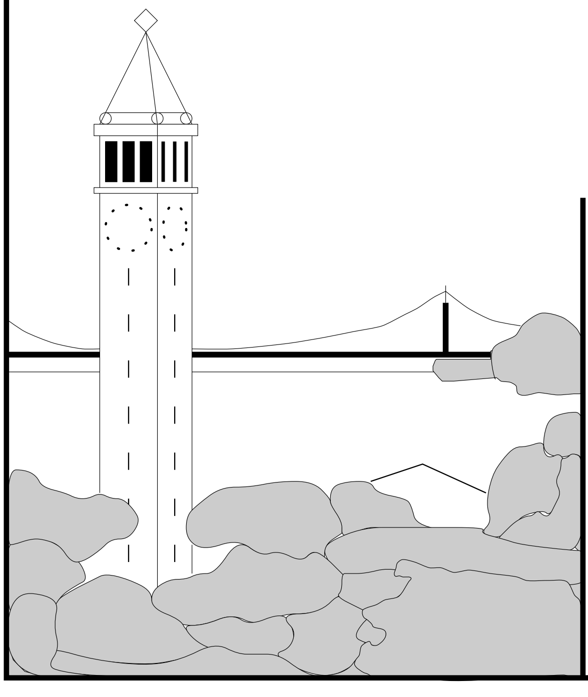
On using Correlation-based Synopses during Query Optimization

Amol Deshpande and Joe Hellerstein



Report No. UCB/CSD-02-1173

February 2002

Computer Science Division (EECS)
University of California
Berkeley, California 94720

This page intentionally left blank.

On using Correlation-based Synopses during Query Optimization

Amol Deshpande and Joe Hellerstein

February 2002

Abstract

In recent years, a variety of complex synopsis structures have been proposed to capture statistical correlations present in database relations. The goal of that work has been to improve cardinality estimation during query optimization by improving statistics on *base tables*. In order to use the correlation information for multi-table estimations as well, synopses must also be constructed on *intermediate result tables* during query optimization. We address that problem here, and show how to efficiently integrate multi-dimensional histograms and other complex synopsis structures into the search algorithm of a traditional query optimizer such as that of System R.

A main challenge in this work is minimizing the computational expense of dynamically computing synopses on intermediate tables during query optimization. We show that one only needs to construct a very small subset of all possible synopses to get the full estimation benefits of multi-dimensional modeling. We also show that choosing this subset unwisely can have unacceptable performance overheads. This leads to an "estimation planning" problem that has been ignored to date in literature: how to choose the most efficiently-computable collection of intermediate synopses to construct, so that all the required cardinality estimates can be computed as accurately as possible from the base-table synopses?

In this paper, we identify and formalize the estimation planning problem in the context of a System R-style optimizer extended with complex synopsis techniques such as multi-dimensional histograms. We propose algorithms that find very good estimation plans efficiently, and discuss properties of synopsis structures that make them amenable to efficient use in an optimizer.

1 Introduction

The query optimization process, which chooses the best plan for executing a query in a database system, typically uses statistical information about the data stored in the system to compare different execution plans. The early systems such as System R [SAC+79], maitained elementary information about the database such as counts of tuples in the tables, the spreads of values for the attributes, the presence or absence of indexes *etc.* This information was then used in conjunction with some simplifying assumptions about the data to compute the information that the query optimizer requires. Two of the most important assumptions that were typically made are :

- *Attribute value independence assumption :* Different attributes in a table are assumed to be independent of each other. For example, suppose we have the query $\sigma_{age \geq 40 \ \wedge \ salary \geq 100000} EMP$ over an *employee* relation. By making this assumption, the optimizer can compute the selectivity of the joint predicate to be the product of selectivities of the individual predicates $\sigma_{age \geq 40}$ and $\sigma_{salary \geq 100000}$.

- *Join predicate independence assumption :* Different join predicates in a query are assumed to be independent of each other. As an example, consider the three relation query as shown in Figure 1(i) and let the selectivities of the join predicates be s_1 and s_2. By assuming that the join predicates, $R.a = S.a$ and $S.a = T.a$ are independent of each other, the query optimizer can estimate the size of the query result to be simply $s_1 \times s_2 \times |R| \times |T| \times |S|$.

These assumptions simplify the computation of cardinality estimates significantly, but rarely hold true in practice [MCS88, PI97]. Most real datasets exhibit strong correlations between attributes that tend to nullify both the assumptions made above. As an example, in an *employee* relation, the attributes *age* and *salary* can be highly correlated, resulting in large errors in the estimates made using these assumptions. Even though the error for a single predicate or for a single join may not be very high, these errors tend to propagate multiplicatively in a multi-way join query [IK90, Col00, Ant93] and can result in very large errors later in the process.

To solve this problem, many complex synopsis techniques have been proposed recently that capture statistical correlations between attributes of a table. These techniques include multi-dimensional histograms [MD88, PI97], Wavelet-based synopses [VWI98], Sampling [SBM93, Wil91], Dependency-based histograms [DGR01] among others. As these techniques model the multi-dimensional distribution over attributes of a table directly, it is quite straightforward to use them to relax the attribute value independence assumption. On the other hand, use of these techniques to relax the join predicate independence assumption requires combining these synopses together to compute synopses on intermediate tables.

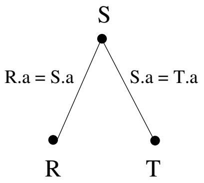
As an example, let the distributions over the attribute a (assumed to take three distinct values) for the three relations R, S and T in our earlier example, be as shown in Figure 1(ii). Given this distribution information, the selectivities of the two joins are $s_1 = 1/3$ and $s_2 = 1/3$. If the join predicates are assumed to independent of each other, then the cardinality of the result relation ($R \bowtie S \bowtie T$) will be estimated (using the formula described earlier) to be 3. But as we can see, the two join predicates are not independent of each other, and the correct number of tuples in the query result is actually 0. This simple example illustrates that the errors made using the join predicate independence assumption can be arbitrarily large.

To relax this assumption, synopses need to be built on the intermediate tables as well. In the above example, building a synopsis on the intermediate table $R \bowtie S$, and then using this synopsis to estimate the selectivity of the other join will show us that there are no tuples in the query result. Unfortunately, for these kinds of complex modeling techniques, the computational costs of building such synopses can be very high. We show that, in general, only a very small subset of possible intermediate table synopses is required to be constructed to compute all the estimates that a traditional query optimizer such as that of System R needs. This leads to an "estimation planning" problem that has been ignored in literature to date : *how to choose the most efficiently-computable collection of intermediate synopses to construct, so that all the required cardinality estimates can be computed as accurately as possible from the base-table synopses?*

In this paper, we identify and analyze this estimation planning problem. We demonstrate that good estimation planning can have optimization-time benefits of orders of magnitude over naive estimation planning techniques. We then provide algorithms to efficiently and effectively perform estimation planning. Though we will focus on multi-dimensional histograms, we also discuss how the algorithms can be extended to a variety of other synopsis techniques.

1.1 Outline of the Paper

We will begin with a brief description of the related work, in particular, complex synopsis techniques that have been proposed to capture attribute correlations, as well as the problem of using



(i)

Value	Count
x	3
y	0
z	0

Relation R

Value	Count
x	1
y	1
z	1

Relation S

Value	Count
x	0
y	0
z	3

Relation T

(ii)

Figure 1: The join predicate independence assumption

these techniques (Section 2). Then, we will formally define the problem and analyze the space of solutions; we also show how estimation planning is integrated into a traditional query optimizer (Section 3). We will develop an exponential-time exhaustive algorithm for solving the problem, and develop heuristics (Section 4) that are much more efficient, yet consistently found very good plans in our experimental study. For brevity, we discuss only multi-dimensional histograms in detail, but we briefly explain how the algorithms we propose can be extended to other synopsis techniques (Section 5). Finally, we present a performance study that validates the claims made in this paper (Section 6).

2 Related Work

There is a large body of work on constructing sophisticated synopsis structures; a survey appears in [BDF$^+$97]. Due to space constraints, we only discuss a few of these techniques in detail here.

[SAC$^+$79] recognized that estimating the sizes of intermediate results was crucial to the success of query optimizers, and proposed using a simple estimation technique based on assuming uniform distributions of attribute values. [Chr83, PSC84] propose using histogramming techniques for better estimation, but they still make the attribute value independence assumption that we outlined earlier. To our knowledge, the first work to consider the effect of correlated attributes is by Muralikrishna and Dewitt [MD88] who propose approximating the multi-dimensional distribution over the attributes of a relation directly. Their technique is very similar to the multi-dimensional histogramming technique proposed by Poosala and Ioannidis [PI97], called MHist. Though the construction and the use of histograms are different for these two techniques, both of them partition the multi-dimensional space to be approximated into disjoint buckets, and assume uniform distribution within each bucket. Partitioning schemes will prove to be important for our discussion of constructing synopses over intermediate relations. Figure 2 shows how an MHist may approximate the joint distribution over two variables.

[VWI98, MVW00] propose using Wavelets to summarize a multi-dimensional dataset. Their approach involves computing a multi-dimensional Haar Wavelet on the data, and storing only the most significant coefficients of this Wavelet as the synopsis. These coefficients can then be used for selectivity estimation, and also for approximate query processing [CGRS00]. Aboulnaga and Chaudhuri [AC99, BCG01] propose a different kind of histogramming technique called *self-tuning histograms* that are built by looking only at the result sizes of the queries that are being executed by the system. In terms of partitioning, both of these histogramming techniques may allow overlapping buckets, something not allowed by the two techniques discussed above.

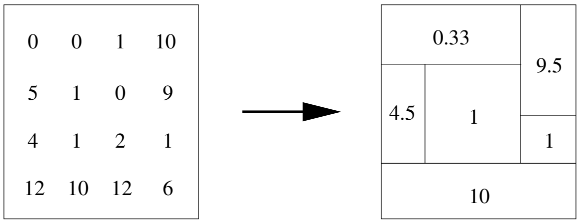


Figure 2: An example MHist

[DGR01, GTK01] propose applying statistical modeling techniques to the problem of selectivity estimation. *Dependency-based histograms* [DGR01] make use of correlations between attributes of a relation to approximate a high-dimensional distribution with a collection of smaller dimensional histograms. The smaller dimensionality histograms can be built using any histogramming technique. [GTK01] propose building a Bayesian network not only over the attributes of a single relation, but across relations through use of foreign-key dependencies. This technique can be used for selectivity estimation as well as for estimating cardinalities of join queries that involve only foreign key joins.

There has been surprisingly little work on actually using these synopsis structures during the query optimization process. Mannoni *et al* [MCS88] address this problem in some detail, but their focus is mainly on *how to* construct synopses on the intermediate tables from the base table synopses rather than how to *efficiently* compute the required cardinality estimates. The computational complexity aspect of the problem is not critical for them as they only consider simple one-dimensional synopsis structures.

The issue of using synopsis structures efficiently also arises in *approximate query processing* [GGgz]. Again, there has been very little work addressing this problem. [DGR01] allude to this problem briefly, but do not propose any concrete ideas to solve it. The algorithms we present in this paper can be extended to efficiently incorporate complex synopsis techniques in approximate query processing as well (Section 7).

3 Problem Definition and Analysis

In this section, we define the estimation planning problem in more detail, and analyze the solution space. We also describe how estimation planning fits into a traditional query processing algorithm. For the sake of exposition, we will focus in this section on multi-dimensional histograms as our synopsis techniques, but the discussion here applies to other kinds of synopsis techniques as well.

3.1 Operations on Histograms

Before addressing estimation planning, we briefly discuss the operations that must be performed on histograms during the estimation process.

We will use the notation H_s^S to denote a histogram on relation S on attribute(s) s. Thus, H_a^A denotes a one-dimensional histogram on relation A and attribute a, whereas H_{ab}^B denotes a two-dimensional histogram on relation B and attributes a and b.

During estimation, we need to manipulate histograms in a manner analogous to the operations on the relations they summarize. Hence there are three operations applied to histograms during the

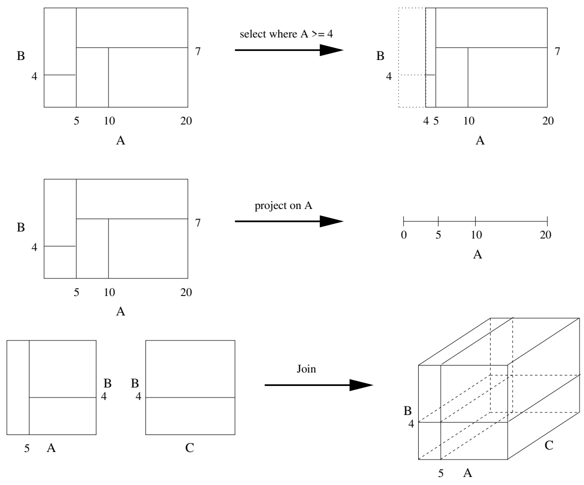


Figure 3: Generation of Intermediate Synopses

estimation process:

- **Selection :** This involves selecting only the part of a histogram that satisfies a given predicate. Usually, only range selection predicates on the relations can be translated into selections on the histograms.
- **Projection :** This operation involves projecting the histogram onto a subset of the attributes on which it is built. We will denote projecting a histogram on relation S and attribute set s onto the attribute set $s' \subseteq s$ by $H_s^S \xrightarrow{project} H_{s'}^S$.
- **Join :** Joining two histograms can be thought of roughly as performing a join on the corresponding relations and building a histogram on the result relation. Clearly that is an inefficient scheme for constructing a join of two histograms. The implementation of histogram-join can be tricky and depends largely on the histogramming technique as we will discuss in Section 5. We will denote this operation using the standard notation for relational join, $\bowtie$.

These operations are analogous to the standard relational operators, but are performed upon and return histograms, not relations. Throughout the course of the paper, when we refer to Selection, Projection and Join, we will be referring to these variants over synopses, and not the traditional operators over relations.

Figure 3 shows some examples of these operations on the MHist histograms that we outlined earlier.

3.2 Incorporating Histograms in the Optimization Process

During the optimization process, the optimizer has to estimate the cardinalities of all the intermediate relations that it considers. Selections and projections on base tables, if any, can be applied to the base table histograms directly to obtain histograms on the result relations using the select and project operations described above. To estimate the size of a two-table intermediate relation, say RS, the optimizer needs to perform a join on the corresponding base-table histograms to obtain a histogram on the table $R \bowtie S$ (denoted as RS from now on), which is then used to estimate the size of the relation. Similarly, to estimate the cardinality of a three-table intermediate relation, say $R \bowtie S \bowtie T$ (RST), the optimizer needs to join the histograms on the base tables, R, S and T, and

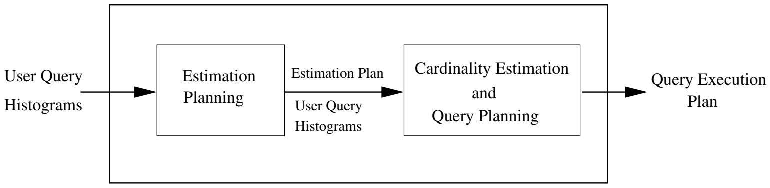


Figure 4: Incorporating Estimation Planning in the Optimization Process

the result histogram is then used to estimate the size of that relation. In general, there are a very large number of ways to compute the required estimates, as the same intermediate relation can be constructed in different ways from the base relations. This leads us to the definition of an estimation plan :

Definition 3.1: *An* estimation plan *is a sequence of select, project and join operations on the synopses that produces estimates for all the intermediate cardinalities that are needed for query optimization; the estimates that are produced are as accurate as possible given the base table synopses*[1]. ∎

As an example, a naive way of doing estimation is to estimate the cardinality of each intermediate relation separately by joining together the corresponding base relation histograms. Clearly, this is not a very efficient estimation plan, as the histograms that have already been built on intermediate tables can be and should be used to compute further estimates. Continuing with our example, the cardinality of the relation RST can be more efficiently estimated by joining the histogram on relation RS that has already been built and the histogram on relation T. This leads us to the problem that we address in this paper :
Estimation planning problem : Given a query and a set of base relation histograms corresponding to the query, find the most efficient estimation plan for computing all the required estimates that the query optimizer needs.

Though Definition 3.1 is quite general, any estimation plan that does not reuse the intermediate table synopses is clearly sub-optimal. From now on, we will restrict our attention to plans that do reuse such intermediate synopses.

In a traditional optimization algorithm, the estimation process itself is performed on demand as new intermediate relations are generated. The estimation planning algorithms we propose in this paper are used as a preprocessing phase that tells the optimizer which intermediate synopses should be constructed for optimal computation of the required estimates (Figure 4). Note that estimation planning doesn't create the intermediate synopses itself, but only decides which synopses to construct. The creation of synopses is integrated into the optimizer's cardinality-estimation routines using the selection, projection and join operations as described in the previous section.

3.3 Problem Definition

We are given a set of relations and a multi-way join query over these relations. For brevity, we will assume that the query only contains join operators. This query is represented in the form of a *query graph*, where the vertices of the query graph correspond to the input relations, and an edge in the

[1] In this paper, we do not allow for approximations during the estimation process. We will revisit this issue in Section 7.

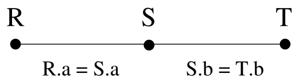


Figure 5: An example query graph

query graph corresponds to a join between the corresponding relations in the query. Figure 5 shows an example query graph over three relations. For each relation in the query, we are given a *multi-dimensional histogram* over all the attributes of the relation that are involved in any of the joins. Given these histograms, we are required to estimate the cardinalities of all intermediate relations that could be generated by a standard System R-style query optimizer[2].

Continuing with our example (Figure 5), the System R query optimizer will require estimating the sizes of the relations RS, ST, and RST from the three base table histograms, (1) H_a^R, (2) H_b^T, and (3) H_{ab}^S. Note that the former two are one-dimensional histograms, whereas the third histogram is a two-dimensional histogram. There are two estimation plans for this query that reuse intermediate table synopses :

1. $H_a^R \bowtie H_{ab}^S \longrightarrow H_{ab}^{RS} \xrightarrow{project} H_b^{RS}$
   $H_{ab}^S \xrightarrow{project} H_b^S$
   $H_b^S \bowtie H_b^T \longrightarrow H_b^{ST}$
   $H_b^{RS} \bowtie H_b^T \longrightarrow H_b^{RST}$
   The cardinalities of the three intermediate relations, RS, ST, and RST are computed as the histograms H_{ab}^{RS}, H_b^{ST} and H_b^{RST} are built. Note that the last two histograms, H_b^{ST} and H_b^{RST}, don't actually need to be built as the estimates can be computed while performing the joins. Though this results in reduction in the amount of memory required during estimation, it does not affect the computational cost of estimation.

2. $H_{ab}^S \longrightarrow H_a^S$
   $H_a^R \bowtie H_a^S \longrightarrow H_a^{RS}$ (estimate $|RS|$)
   $H_{ab}^S \bowtie H_b^T \longrightarrow H_a^{ST}$ (estimate $|ST|$)
   $H_a^R \bowtie H_a^{ST} \longrightarrow H_a^{RST}$ (estimate $|RST|$)
   For clarity, we do not show the project operations explicitly here; they can be inferred from the subscripts.

As we can see, the histogram H_b^T is involved in two join operations in the first plan, but only once in the second plan, whereas H_a^R is joined once in the first plan and twice in the second plan. If the sizes of these two histograms differ considerably, then even in this simple example, one plan can be significantly more cost-effective than the other plan.

3.4 Solution Space

Planning problems are only difficult if the solution space is very large; that is indeed the case here. Consider a n-relation join query without any selection predicates. Depending on the shape of the query graph, the number of intermediate relations varies between $O(n^3)$ (for a "path query") and 2^{n-1} (for a "star query") [OL90]. Now, the cardinality estimate for any intermediate relation r_i can be computed by building histograms on any two (possibly intermediate) relations whose join

[2]Though we assume that the query optimizer uses a System R-style dynamic programming algorithm (extended to search through bushy plans), the discussion in this paper can be easily extended to other commercial optimization techniques like the top-down scheme of Cascades used in products like MS SQL Server [Gra95]

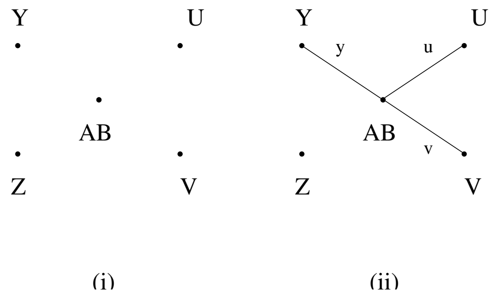


Figure 6: The relations for which cardinalities can be estimated (EST) for different choices of $X \subseteq \{x, y, u, v\}$; (i) $X = \phi, EST = \{AB\}$; (ii) $X = \{y, u, v\}, EST = \{Join(S)|\{A, B\} \subseteq S \subseteq \{A, B, Y, U, V\}\}$

produces r_i. For example, for the query graph of Figure 5, the cardinality of the relation RST can be computed either from histograms on relations R and ST, or from those on relations RS and T. Note that the choice of histograms to be used for estimating the cardinality of some intermediate relation is independent of the choice made for other intermediate relations. This results in an explosion in the number of possibilities for computing the required estimates. For a star query with n relations, this number can be (very conservatively) estimated to be $O(2^{2^n})$, whereas for a n-relation path query, there are at least $O(n^{n^2})$ solutions to this problem (Please refer to Appendix B for a proof of this claim).

3.5 A Constrained Solution Space

In this section, we show how to constrain the solution space so that it is easier to describe the solutions and search in this smaller space. As we will show, this constraint arises naturally, and in many cases the optimal solution should be present in this space. For sake of clarity, we will temporarily assume that the query graph is a tree; we show how to relax this assumption in Section 3.6.

Let T be the given query tree, and let $A \bowtie B$ be a 2-relation join in this query. Let $H^A_{S_a}$ and $H^B_{S_b}$ be the histograms on these two relations on attribute sets S_a and S_b respectively. During the course of estimation, we will have to join these two histograms at some time to estimate the cardinality of relation AB, as that is the only way we have available to compute the cardinality of this relation. As a result of this join, we get a histogram $H^{AB}_{S_a \cup S_b}$ on the intermediate relation AB. Now, during the estimation process, we might want to store a histogram on this intermediate relation for use in further estimation. This stored histogram could be any projection of $H^{AB}_{S_a \cup S_b}$, including H^{AB}_{ϕ} – *i.e.*, no histogram is stored for this relation.

The key observation to make here is that projecting out attributes that participate in further joins limits the use of this stored histogram for subsequent estimation. Suppose we form the projection H^{AB}_X, and later want to estimate the cardinality of the result of joining AB with another relation R. If X does not include the attributes that join AB with R, then we must compute the cardinality of ABR using some other histograms (e.g. H^{BR} and H^A).

This notion can be formalized into a simple conceptual procedure to find all the intermediate relations for which the cardinality can be estimated using H^{AB}_X and the rest of the base relation histograms :

- Merge the two nodes A and B in the query graph T to form a new node AB. All the edges

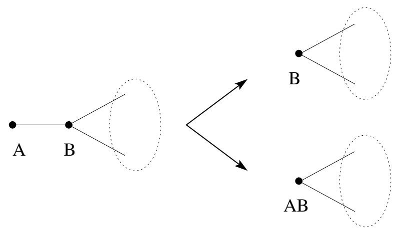


Figure 7: Collapsing a leaf edge

incident on A and B will now be incident on AB (except for (A, B)).

- For every attribute z that is *projected out*, *i.e.* for every $z \notin X$, remove the edge corresponding to that attribute from T (Duplicate join attributes, if present, can be renamed temporarily so that each attribute corresponds to exactly one edge in the query graph).
- Enumerate the intermediate relations that can be generated by the connected component of T that contains the relation AB. These are the only relations for which the cardinalities can be estimated using just H_X^{AB} and histograms on base relations other than A and B.

Figure 6 shows an example of how the choice of X determines the intermediate relations for which the cardinalities can be estimated. $Join(S)$ denotes the intermediate relation formed by joining the relations in S.

The above discussion immediately leads us to the following theorem :

Theorem 3.1 *Given a query graph T and a estimation plan P, there exists at least one edge $e = (U, V) \in T$ such that the entire multi-dimensional histogram that is the result of joining U and V is materialized in P.*

Proof: Intuitively, if the entire result histogram is not materialized for at least one join present in the query, then there is no way to estimate the cardinality of the overall query result accurately. ∎

Definition 3.2: *Given an edge $e \in T$, the process of materializing the entire result histogram that is the result of joining the two endpoints of e is called* collapsing edge *e.* ∎

Given this definition, we make the following claim :

Claim 3.1 *In an optimal estimation plan, if the edge (A, B) is collapsed, and the resulting histogram is used to estimate the cardinality of the overall query result, then this histogram is also used to estimate the cardinalities of all intermediate relations that contain both A and B.*

The intuition behind this claim is that, if the best way of estimating the cardinality of the query result involves collapsing the edge (A, B), then it must also be the best way to compute cardinality of any intermediate relation that contains both A and B. We will assume that this claim holds true for rest of the paper, even though we have not been able to prove it in general. As we will see in our experimental results, in spite of making this assumption, we are still able to find very good estimation plans.

Given this claim, we have that if the edge connecting relations A and B is collapsed, then the resulting histogram on AB is used to estimate the cardinalities of all the intermediate relations that contain AB. We also need to estimate the cardinalities of the remaining intermediate relations. Both

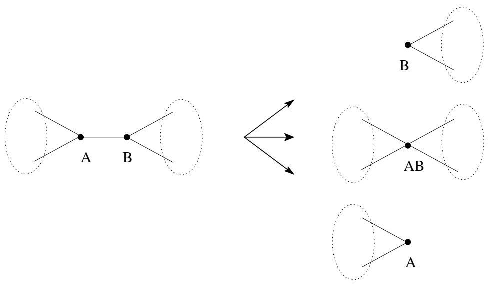


Figure 8: Collapsing an interior edge

of these can be achieved by *recursively solving* multiple smaller sized subproblems. The structure of the subproblems created depends on whether the edge that is collapsed is a leaf edge or an interior edge.

- **Leaf Edge :** As shown in Figure 7, collapsing a leaf edge (A, B) results in two subproblems :
  - A subproblem on the query graph obtained by merging the two nodes A and B. The histogram on node AB is formed by joining the histograms on relation A and B. The join attribute is projected out from H^{AB} if it is not needed further.
  - A subproblem on the query graph obtained by deleting the node A and the edge (A, B). Again, the join attribute corresponding to the leaf edge can be projected out from H^B if not needed further.

  The first subproblem is to estimate the cardinalities of the intermediate results that include both the relations A and B, whereas the second subproblem is to estimate the cardinalities of the rest of the relations. The rest of the histograms from the original problem are passed unmodified to both the subproblems.

- **Interior Edge :** Collapsing an interior edge (A, B) results in three subproblems as shown in Figure 8.
  - Subproblem formed by merging the two nodes A and B and joining the histogram on these relations to get a histogram on relation AB.
  - Two subproblems obtained by removing the edge (A, B) from the original query graph.

  Note that, the histogram on the relation AB will be larger in dimensionality that either H^A or H^B, even if the join attribute is projected away immediately.

The subproblems that are created after collapsing an edge, if solved independently of each other, may *both* estimate cardinalities of some intermediate relations (specifically, the intermediate relations that do not contain either A or B). For example, Figure 9 shows the result of collapsing the edge between relations A and B in an example query graph. As we can see, both the subproblems share the computation of estimating the cardinality of relation CD. While executing such an estimation plan, these computations are not repeated. We will refer to these as separate subproblems for sake of clarity.

As we assume that the Claim 3.1 is true, this subspace of estimation plans contains the optimal estimation plan, and as such we will focus only on this subspace of plans in the rest of the paper. Note that in spite of imposing this constraint on the plans considered, this space is still very large. For a star query with n relations, the total number of plans is still more than $O(2^{2^n})$.

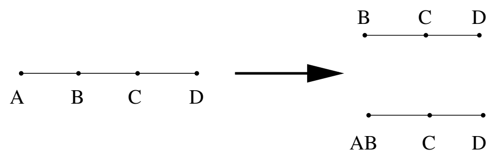


Figure 9: Subproblems after collapsing edge (A, B) both compute cardinality of relation CD

3.5.1 Leaf-only Plans

The difference in the subproblems that are created based on whether the edge that is collapsed is a leaf edge or not leads to a natural subclass of estimation plans that only collapse a leaf edge at any stage. We will call these kinds of estimation plans *leaf-only plans*. These plans form a very important subspace of estimation plans. As discussed earlier, collapsing an interior edge always creates a histogram larger in dimensionality that any of the input histograms. As such, an optimal estimation plan rarely involves collapsing interior edges and the optimal plan can be expected to be often found in the space of leaf-only plans.

3.6 Cyclic Queries

Cyclic queries are those in which the query graph has cycles. There are two possible scenarios here based on whether the optimization algorithm itself makes use of all the joins in the query :

- **Optimizer breaks cycles :** A common way of dealing with cycles in the query graph is to break the cycles to get an acyclic graph. The join conditions that have been eliminated are then applied at the end as selections. Typically, a minimum weight spanning tree of the query graph, where the weights on the edges are the selectivities of the corresponding joins, is used for this purpose [KBZ86]. In that case, the discussion above and the algorithms we develop later can be applied to the resulting query tree without any modifications.
- **Optimizer does not break cycles :** The other way to handle cyclic queries is to optimize over the entire space of plans that could be generated by any spanning tree of the query graph. In that case, the estimation plan needs to estimate the cardinalities of intermediate relations that could be generated by any spanning tree. We will show how the discussion above can be extended to this case with an example. Consider the query graph as shown in Figure 10. Theorem 3.1 is true for this case as well. Accordingly, let (A, B) be an edge that is collapsed. Then, again assuming that Claim 3.1 holds, the cardinalities of all intermediate relations that contain both relations A and B will be estimated using the resulting histogram. This gives rise to multiple subproblems as shown in Figure 10, that need to be solved recursively. In general, there are three subproblems that can be found as follows :
  - Merge the nodes A and B to get another cyclic query.
  - Delete the node corresponding to relation B to get another (possibly cyclic) query.
  - Delete the node corresponding to relation A.

  As before, these subproblems can share computation and this computation is not repeated during execution of the estimation plan.

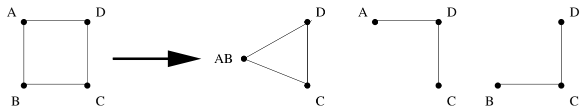


Figure 10: Subproblems generated for a cyclic query

4 Algorithms

In this section, we develop algorithms for finding an estimation plan for a given query graph. All the estimation plans found by our algorithms reside in the constrained solution space introduced in the previous section. We will first present an algorithm that exhaustively searches the constrained solution space in $O(n2^n)$ time and finds the optimal plan under a condition that is satisfied by many histogram techniques. As the computational complexity of the exhaustive algorithm makes it infeasible in general, we present heuristic algorithms that are much more efficient. As we will show later (Section 6), our heuristics produce reasonably good plans in most cases.

4.1 Partitioning Structure Equivalence Property

Definition 4.1: *The* partitioning structure *of a histogram is defined to be the way the histogram partitions the space it approximates.* ▮

As an example, for MHist-style histograms, it is defined by the boundaries of the buckets that the histogram contains. For Wavelets, it is defined to be the supports of the coefficients that are stored, including the sign information [CGRS00]. The actual frequencies stored with the buckets are not relevant to the partitioning structure. The partitioning structure of any synopsis technique is very important, as the computational cost of *using the synopses* is completely determined by the partitioning structures of the synopses.

This observation leads to a large reduction in the number of estimation plans that need to be considered. We formalize this notion through the following property : **Partitioning structure equivalence property :** *Given a histogram H_S^R on relation R and attribute set S, the following two histograms have exactly the same partitioning structure :*

- $H_{S'}^R$*, where* $S' \subset S$ *and* $|S'| = |S| - 1$
- $H_{S'}^{R'}$*, where* $R' = R \bowtie R''$*, for some relation* R''

In other words, the histogram resulting from projecting out a single attribute, and the histogram resulting from joining R with some other relation on that attribute and projecting out that attribute, have exactly the same partitioning structure. Going back to our earlier example (Figure 5), this property means that histograms H_b^S and H_b^{RS} have exactly the same partitioning structure, though the actual frequencies assigned to the buckets in these two histograms will usually be different.

This property is satisfied for all synopsis techniques we are aware of, including MHists, Wavelets etc. Given this property, following theorem holds :

Theorem 4.1 *Given the* partitioning structure equivalence property*, any sub-problem generated using repeated application of* edge collapsing *such that only edges associated with unique join attributes are collapsed, is completely specified by the attributes associated with the edges in the graph corresponding to the subproblem.*

Proof: We will show this in three steps :

- Given a set of attributes S corresponding to a subproblem, we can determine which attribute sets the input histograms are on. This can be done by first collapsing all the edges corresponding to the attributes not in S in the original query graph, and then enumerating the attribute sets corresponding to the nodes in this new graph.
- This is the key step in this proof. Now, if the join attribute associated with the collapsed edge is not associated with any other edge, then this join attribute can be projected away immediately from all the histograms in the resulting subproblems. Given that the partitioning structure equivalence property holds, it is easy to see that the partitioning structure of any histogram in any of the subproblems is completely specified by the attributes on which the histogram is built. As such, the partitioning structures of the input histograms corresponding to the subproblem are uniquely determined.
- For computational complexity purposes, only the partitioning structures of the input histograms are relevant. The actual frequencies associated with the histograms are irrelevant.

Hence, given a set of attributes, there is a unique subproblem that is associated with it.

∎

This theorem proves very useful in searching through the constrained space of estimation plans. For example, the two subproblems that are created after collapsing a leaf edge have the same set of join attributes associated with them, and as such, the optimal estimation plan for solving one of the subproblems is exactly same as the optimal estimation plan for the other subproblem.

4.2 Exhaustive Algorithm

Theorem 4.1 immediately provides us with an exhaustive algorithm that searches through the constrained space of estimation plans exhaustively and finds the optimal plans efficiently as long as there are no duplicate join attributes. We will discuss how this algorithm can be used in case there are duplicate join attributes in Section 4.3.

This recursive algorithm essentially memorizes the solutions for all subproblems in a table indexed by subsets of attributes. The algorithms works as shown in Figure 11.

There are exactly 2^{n-1} possible subsets of attributes for a n relation query, and hence this algorithm runs in $O(n2^n)$ time. Note that this algorithm runs in exponential time even for path queries for which the number of intermediate relations itself is polynomial in n ($O(n^3)$) [OL90].

4.2.1 Cost of Collapsing

One of the key requirements of this, and the other algorithms we develop, is the ability to compute the cost of collapsing an edge (line 8). As described in Section 3.5, this cost consists of performing one or two projections and one join. In general, it is not possible to compute this cost without constructing the resulting histograms themselves, but there are several important cases where it can be estimated accurately. Otherwise, the algorithm will have to be run with approximate costs. We will revisit this issue in more detail when we discuss how to apply these algorithms to prevalent histogramming techniques (Section 5).

4.3 Duplicate Join Attributes

Theorem 4.1 is not satisfied if the join attribute associated with the edge appears multiple times in the query. This problem can not be solved by renaming the join attributes. As an example, consider the

---

Input: $\mathcal{T} = (V, E)$, a join graph; $\mathcal{H}$, histograms on the relations in $\mathcal{T}$; $\mathcal{M}$, a table indexed by sets of attributes that contain optimal solutions as well as their costs (initialized to be empty in the beginning)
Output : Optimal cost of estimating cardinalities for intermediate relations that can be generated by $\mathcal{T}$
begin
$S \longleftarrow$ set of join attributes present in $\mathcal{T}$
if $\mathcal{M}$ contains a solution for S, return the cost
else
  for each edge $e \in E$, **do**
    collapse e to get smaller sized subproblems
    $c_1 \longleftarrow$ sum of optimal costs for the subproblems (found recursively)
    adjust c_1 by subtracting the cost of repeated subproblems
    $c_2 \longleftarrow$ cost of collapsing this edge (the cost of performing required joins and projections)
    $c_e \longleftarrow c_1 + c_2$
  done
  let $e' \in E$ be the edge with minimal total cost
  update $\mathcal{M}$ by adding a new entry $\{S, c_{e'}\}$
  return $c_{e'}$
end

Figure 11: The Exhaustive Algorithm

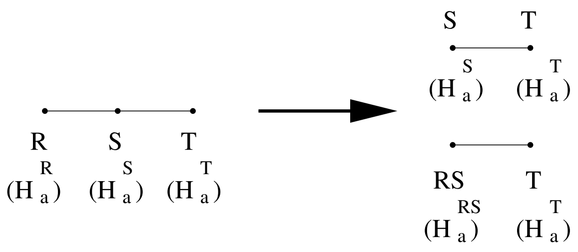


Figure 12: Duplicate Join Attributes

query graph as shown in Figure 12. Suppose the edge (R, S) is collapsed resulting in two subproblems as shown. The first subproblems has an input histogram on relation RS, $H_a^{RS} = H_a^R \bowtie H_a^S$, whereas the second subproblem has as an input the histogram H_a^S. Both these subproblems have the same set of join attributes associated with it, even if we rename the join attributes uniquely. But clearly, the two histograms H_a^{RS} and H_a^S will not, in general, have the same partitioning structures. As such, Theorem 4.1 does not hold in this case, and the exhaustive algorithm can not be used directly.

We get around this problem by never collapsing edges that are associated with duplicate join attribute until the very end when no other edges are present in the subproblem. At that point, the subproblems that need to be solved usually involve low-dimensional histograms (usually not more than one dimensional) and as such, collapsing the rest of the edges in any arbitrary order is not computationally expensive.

4.4 Heuristic Algorithms

The exhaustive algorithm described above runs in exponential time regardless of the query graph shape, and as such, is not feasible in practice. Here we will develop some simple intuitive heuristics that are highly efficient and discuss the circumstances under which they can be expected to work well.

1. **Exhaustive Search over Leaf-only Plans (EXH-LO):** Instead of using the exhaustive algorithm to search the entire constrained estimation plan space, this algorithm only searches through leaf-only plans. The complexity of this algorithm is much lower; in fact, it is comparable to the cost of basic query optimization algorithm. We establish this correspondence through the following theorem (Please refer to Appendix B for the proof) :

   Theorem 4.2 *There is a one-to-one correspondence between the subproblems encountered during an exhaustive search over leaf-only plans and the subproblems encountered in a System R-style query optimizer.*

   As we have argued before, the cost of collapsing an interior edge is, in general, very high compared to the cost of collapsing a leaf edge, and as such, the optimal estimation plan rarely involves collapsing an interior edge for any subproblem. Hence, this algorithm usually finds an optimal estimation plan. Note that this algorithm still takes an exponential time to finish under many query graph shapes such as star queries.

2. **Choose the edge with minimum/maximum cost of collapsing (MIN/MAX) :** As the goal of an estimation plan is to minimize the total cost of collapsing over all collapsed edges, a heuristic that greedily collapses the edge that requires minimum cost of collapsing (MIN) can be expected to perform well. On the other hand, a heuristic that greedily collapses the edge that requires maximum cost of collapsing (MAX) would be expected to do much worse. As it turns out (Section 6), this is not necessarily true, and in fact, MIN results in much slower estimation plans than MAX for some classes of histogramming techniques.

   This counter-intuitive observation can be partially explained as follows : An edge that is collapsed in the beginning is only collapsed once during execution of the estimation plan, and as such the cost of collapsing that edge is only experienced once. On the other hand, the edges that are collapsed later in the estimation plan tend to contribute multiple times to the overall cost of executing the estimation plan as the later subproblems are executed multiple times with different input histograms during the estimation. We will revisit this issue in Section 6.

5 Analysis of Prevalent Histogram Techniques

In this section, we look at various synopsis techniques and analyze the estimation planning problem for these cases. We will discuss the MHist technique in detail as it shares a lot of properties with the other complex synopsis techniques and then briefly discuss the other synopsis techniques.

5.1 MHists

MHists [PI97] approximate the distribution by partitioning the space to be approximated into disjoint buckets recursively. Figure 13(i) shows an example of a 2-d MHist. The numbers on the splits show the order in which the splits were done. MHists are easy to construct and maintain, and have been proven to be highly effective for selectivity estimation [PI97].

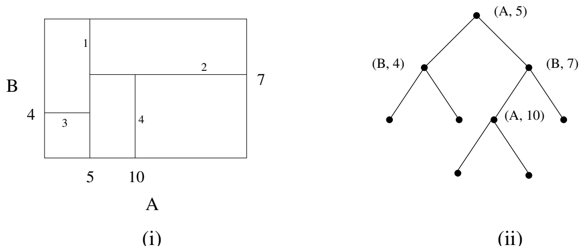


Figure 13: An example MHist

5.1.1 Representation of MHists

There are two ways to represent base relation MHists and the intermediate MHists that are generated in the optimization process.

- **Representation using Bucket Boundaries (BB) [PI97]:** In this representation, the bucket boundaries of the buckets are explicitly represented along with the average frequencies. Though the base relation histograms require the buckets to be disjoint of each other, we will allow for overlapping buckets in this representation.

- **Representation using a split-tree (ST) [DGR01]:** This highly efficient representation stores the splits that were made during construction of a MHist as a tree. The average frequencies are stored at the leaves of this tree. This representation is almost $O(d)$ times more space-efficient that the representation above, where d is the dimensionality of the histogram. Querying is also faster using this representation as the split-tree can be used to guide the search to the leaves that contain the answer. Figure 13(ii) shows the split-tree for the example above.

In general, we expect the base relation histograms to be in the latter format due to low storage requirements, but during the query optimization process, either one of the representations may be used for storing the intermediate MHists that are generated.

5.1.2 Cost of collapsing

As we would expect, the cost of collapsing depends on the representation used for storing the intermediate histograms. We will assume that the base relation histograms are in the split-tree format, which can be easily converted to the other representation if required.

- **BB-representation :** As we allow overlapping buckets, the projection operation is simplified in this representation. It can be achieved by "forgetting" the information about the boundaries along the attributes that are projected out. Join operation is more expensive as we have to compare each bucket from one input histogram to each bucket from the other input histogram to see if there is an overlap on the join attribute. The cost of this is $O(n_1 n_2)$ where n_1 and n_2 denote the numbers of buckets in the input histograms. Note that it is not possible to predict how many buckets are going to be in the result histogram without performing the join itself, unless other auxiliary information is maintained with the histograms.

  We identify one important special case in which it *is* possible to predict the number of buckets in the result histogram. When one of the input histograms is one-dimensional and the join attribute is immediately projected away, then the number of the buckets in the result histogram is exactly same as the number of buckets in the other histogram. Note that, this is the only join operation that is performed if the *distinct join attributes assumption* is satisfied, and we consider only leaf-only estimation plans.

- **ST-representation :** [DGR01] describe how join and projection operations can be performed on MHists represented as split-trees so that the output MHist is also in the same format. For sake of completeness, we describe these algorithms in Appendix A. The cost of the project operation can be approximated as the size of the resulting histogram, whereas the cost of a join operation is always less than $n_1 n_2$, but it is not possible to predict it more accurately. A join operation costs less in this representation because the split-tree representation is used to eliminate comparisons between buckets that are definitely non-overlapping. In general, it is not possible to predict the result sizes of either of these operations. The requirement that the result be in split-tree format results in a large variability in the result sizes. This variability also makes it imperative that we be able to at least estimate the result sizes as choosing a bad edge to collapse could result in very large intermediate histograms. We estimate this number under worst case assumptions by making just one bottom-up pass on the tree. Briefly, for each node in the subtree, we count the number of splits along each dimension in that subtree. If the node involves split on an attribute that is being projected away, these two vectors of split counts are combined by assuming that this projection creates a regular grid on the space covered by the subtree. This generates a very conservative estimate of the size of the projection, but we found that it worked very well in practice. In future, we plan to look into this issue more closely.

We will revisit the issue of which representation should be used during the query optimization process in Section 6.

5.2 Other Synopsis Techniques

We will briefly analyze three other synopsis techniques with regards to applicability of the cost models and algorithms discussed above.

- **Wavelet-based Synopses:** As regards to computational complexity, Wavelet-based synopses [VWI98, MVW00] are very similar to the MHist histograms discussed above. In fact, [CGRS00] propose storing these synopses by explicitly recording the bucket boundaries. The main difference between these two techniques is that a given Wavelet coefficient may contribute different values to the space associated with it. Though the absolute value contributed is the same, it may contribute a negative value to some regions of the space, and positive to the other. In spite of this difference, the cost models and the algorithms described above can be easily extended to this case.
- **Self-tuning Histograms [AC99, BCG01] :** Though Self-tuning histograms are unique in that they are constructed without looking at the data, with regards to computational cost of histogram operations, these are also very similar to the MHist histograms.
- **Dependency-based Histograms [DGR01] :** These histograms use statistical modeling techniques to discover dependencies between attributes of a table and use these dependencies to approximate a high-dimensional probability distribution with a collection of smaller dimensionality histograms. These histograms pose an additional degree of complexity as the smaller histograms have to be combined at runtime to get a histogram on the base table. Unfortunately, the approach of combining the smaller histograms associated with each base table separately to get one histogram per table can result in very large intermediate histograms. The

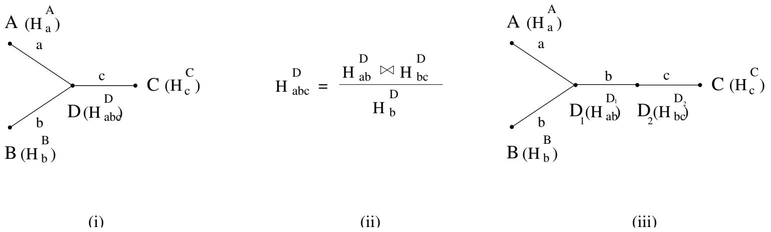


Figure 14: (i) Example query graph; (ii) Decomposition of H_{abc}^D; (iii) New query graph

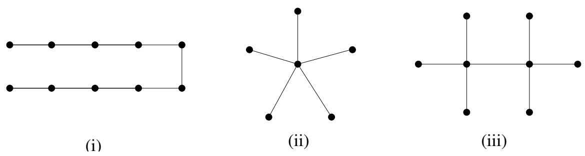


Figure 15: Query graph shapes; (i) path query, (ii) star query, (iii) two stars query

algorithms we present in this paper can be used instead to avoid construction of such large tables.

We will show this through an example. Consider the query graph as shown in Figure 14(i) and let the decomposition of the three dimensional histogram be as shown in Figure 14(ii). Composing these two smaller histograms on the relation D to get a histogram on all the attributes involves an operation very similar to a join, though with a slight difference as described in [DGR01]. Given this kind of decomposition, we can create a new problem to be solved as shown in Figure 14(iii). Essentially, all the external edges are associated with the smaller dimensional histograms depending on the corresponding join attributes. The edge (B, D) can be attached to either of the two histograms on D. Now, the algorithms described in this paper can be used to solve this new problem. Note that, this new problem will find the cardinalities of many relations such as AD_1 that don't exist and these cardinalities are thrown away. Only cardinalities of those relations that contains both D_1 and D_2 are required. Note that, the join involving the edge between D_1 and D_2 is a slightly different kind of join as mentioned before. This is of no consequence during the estimation planning as the cost of this join operation is similar to the cost of histogram-join, but during estimation, correct join should be used.

Note that, this new problem will almost always contain duplicate join attributes. Though the algorithms described in this paper can still be used to solve the new problem, we plan to look into the case of duplicate join attributes in more detail in future.

6 Performance Study

In this section, we will present a performance study evaluating the various algorithmic techniques that we described in this paper. We will concentrate on the MHist-style histograms and discuss both the BB and ST representations in detail as they present two extremes in complexity of operations and as such, cover a large number of other synopsis techniques (Section 5).

The main results of this section can be summarized as follows :

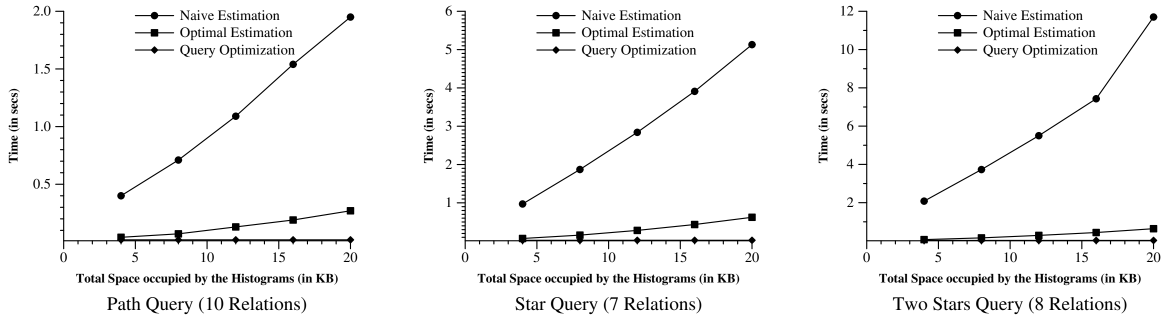


Figure 16: Estimation Times for Naive and Optimal Plans, and Query Optimization Time

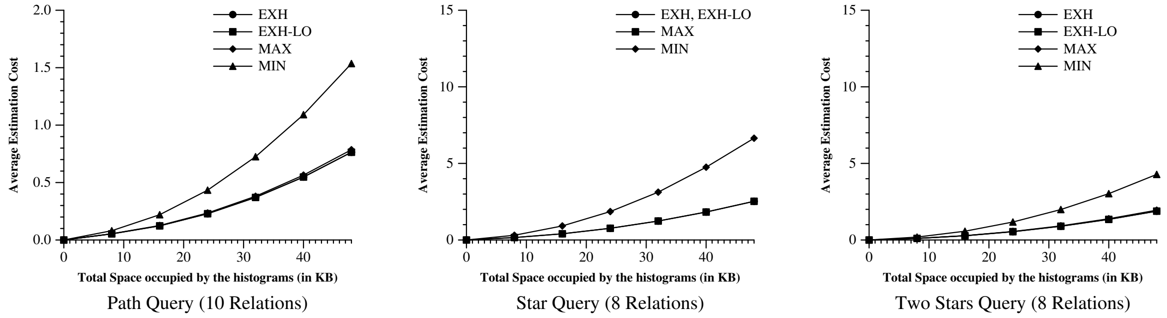


Figure 17: Performance of the algorithms (BB Representation)

- The cost of estimating the intermediate table cardinalities can be very high, especially for complex histogramming techniques.
- The heuristic algorithms proposed in this paper find good estimation plans very efficiently.
- The cost of estimation planning, *i.e.*, the cost of finding a good estimation plan, is more than offset by the savings obtained by using a good estimation plan during estimation.

6.1 Experimental Setup

As the critical input to the estimation planning problem is the shape of the query graph, we ran experiments for various different query graph shapes. We report results from three query graph topologies: (1) path queries, (2) star queries and (3) queries with two stars (Figure 15). As the cost of computing the cardinality estimates only depends on the partitioning structures of the histograms, we generate the base relation MHists randomly as required, so that we have flexibility in choosing the sizes of histograms as well the query topologies. All the experiments were conducted on a dual Pentium 667MHz machine running Linux with 512 MB of RAM.

6.2 Cost of Estimation

With this simple experiment, we show that the cost of computing the required cardinality estimates can be very high, especially if the estimation plan is not chosen intelligently. We will compare the estimation times of the naive estimation technique that estimates the cardinality of each intermediate

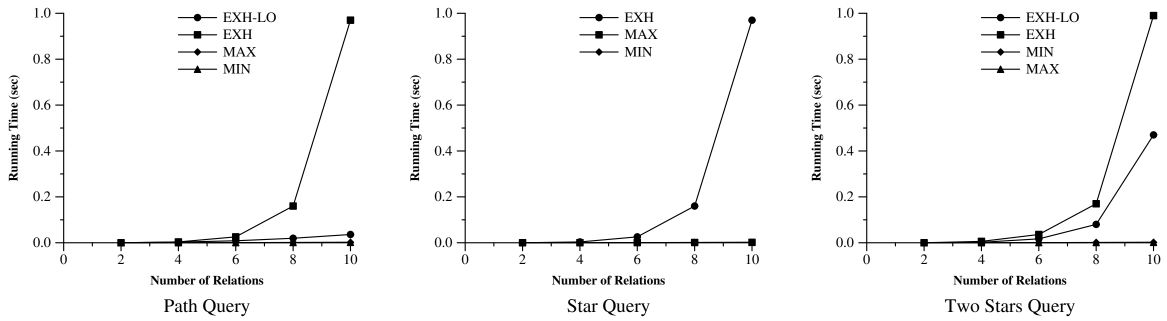


Figure 18: Running Times of the algorithms (BB Representation)

relation separately using just the base relation histograms, and the optimal estimation plan that reuses the intermediate synopses. We will only show this for the BB-representation MHists, as the results are very similar for the ST-representation MHists.

For this experiment, all the histogram sizes were chosen to be equal and the total space occupied by the histograms was varied from 4KB to 20KB. This is actually smaller than the typical sizes of synopses in current database systems, but the naive estimation plan does not finish in reasonable time with larger histogram sizes. For comparison purposes, we also show the running time of the basic query optimization algorithm assuming that computing estimates takes zero time. As we can see in Figure 16, there is an order of magnitude difference between the naive plan and the optimal plan, especially for the two stars query topology. The query optimization time is smaller than either of these two estimation times, and barely registers on the graph.

6.3 BB-representation

We will first show the performance of the algorithms as well as the running times of the algorithms for the BB-representation.

6.3.1 Performance of the Algorithms

With this experiment, we will compare the effectiveness of the algorithms that we developed in this paper in finding a good estimation plan. We compare the three heuristics that we developed in Section 4.4 with the exhaustive algorithm. The sizes of the histograms were randomly chosen so that the total space occupied by the histograms is constant. We show the running times of the estimations plans chosen by the algorithms averaged over 10 runs. As we can see in Figure 17, the optimal plan is almost always found in the space of leaf-only plans. Because of the high cost of collapsing an interior edge, such edges are rarely collapsed in favor of leaf edges. For star queries, all the plans are leaf-only plans and hence we do not show different curves for these two algorithms in that case. The MAX algorithm also tends to find very good estimation plans, whereas the other greedy heuristic, MIN, finds much worse plans. In fact, for the star query topology, the MAX algorithm can be shown to be optimal for these kinds of histograms.

This performance of the greedy heuristics is consistent with the argument we outlined in Section 4.4. The edges that are collapsed earlier tend to contribute only a few times to the overall cost of estimation, whereas the edges that are collapsed later appear in multiple subproblems and as such, tend to contribute multiple times. As such, a heuristic that collapses the minimum cost edges

Original No. of Buckets	No. of buckets in projected MHists
100	94, 152, 179, 679, 833, 916
200	410, 422, 619, 2806, 3066, 3331
500	1553, 1860, 2454, 15229, 17671, 23394
1000	5438, 6947, 11644, 57428, 61757, 101421
2000	18869, 22739, 48573, 190104, 246900, 463324
5000	93011, 152519, 242515, 945354, 1702143, 2537899

Table 1: Explosion after performing a projection on the ST-representation

greedily can be expected to perform worse than a heuristic that collapses the maximum cost edges first.

6.3.2 Running Time of the Algorithms

Figure 18 shows the running times for these algorithms for the three query topologies. None of the algorithms are affected by actual sizes of the histograms, and as such, the complexity of each algorithm only depends on the query shape. As we can see, for the path query topology, the running time of EXH-LO is much lower than that of EXH. This is because the former is a $O(n^2)$ time algorithm, whereas the latter is an exponential time algorithm. For star queries, all plans are leaf-only and as such both the algorithms take same time to finish. For two stars topology, the difference between the two algorithms can be proved to be about a factor of 2 as can also be seen in the corresponding graph. Both the greedy heuristics run much faster than the other algorithms, as both are them are essentially linear time algorithms.

The benefits of intelligent estimation planning can easily be seen from these two experiments. The difference between a good estimation plan and a bad estimation is much larger than the running time of the MAX algorithm, and even the EXH-LO algorithm in many cases. Considering that the total space allocated to histograms in typical database systems is quite large, the benefits of estimation planning can be tremendous.

6.4 ST-representation

The behaviour of the estimation planning algorithms changes considerably if the intermediate histograms are stored in the ST-representation or any representation that does not allow overlapping buckets. The main reason behind this is the requirement of keeping the resulting histograms in non-overlapping buckets format results in a large number of buckets in the histograms.

We illustrate this phenomenon through an example. Table 1 shows the number of buckets in the projections of a 6-dimensional MHist on all 5-attribute subsets of the 6 attributes. As we can see, the number of buckets in the projection can be much larger compared to the number of buckets in the original MHist and also, the projection seem to exhibit a large variability in the sizes. While underlining the importance of choosing an estimation plan correctly, this also begs the question of why the intermediate histograms should be stored in this format at all. We will address this question in Section 6.5.

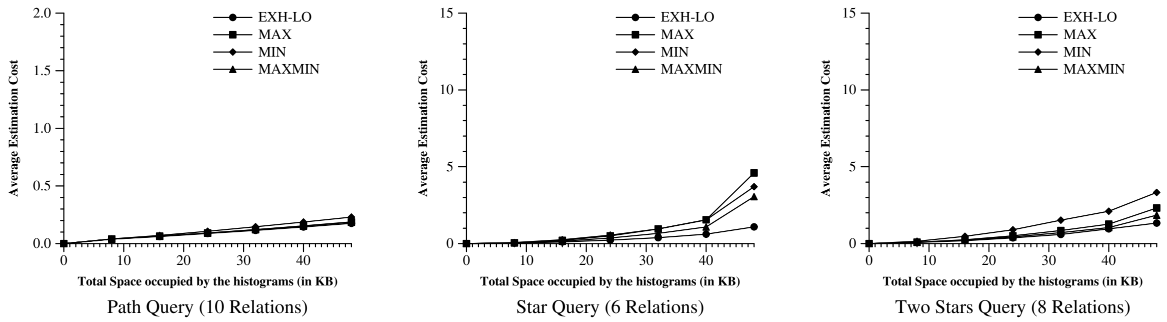


Figure 19: Performance of the algorithms (ST Representation)

6.4.1 Performance of the Algorithms

Figure 19 shows the performance of various estimation planning algorithms in this scenario. The experimental setup is the same as above. We do not show the results for EXH as the algorithm does not run in reasonable time except in very small cases. The main problem is that collapsing an interior edge results in a histogram that is larger in dimensionality that either of the input histograms, and in general, has much larger number of buckets than either of them.

As we can see, both the greedy heuristics MAX and MIN performs much worse than the exhaustive algorithm EXH-LO. We have already seen the intuition behind bad performance of the MIN algorithm, but even the MAX algorithm performs badly in this case. This is because while choosing to collapse edges that have maximum cost of collapsing, this algorithm generates intermediate histograms with very large sizes. For example, returning to our earlier example (Table 1), MAX algorithm may choose to collapse the edge that results in creation of the histogram with size **2537899**. This problem does not arise in the BB representation because all the projections of a given histogram contain exactly the same number of buckets.

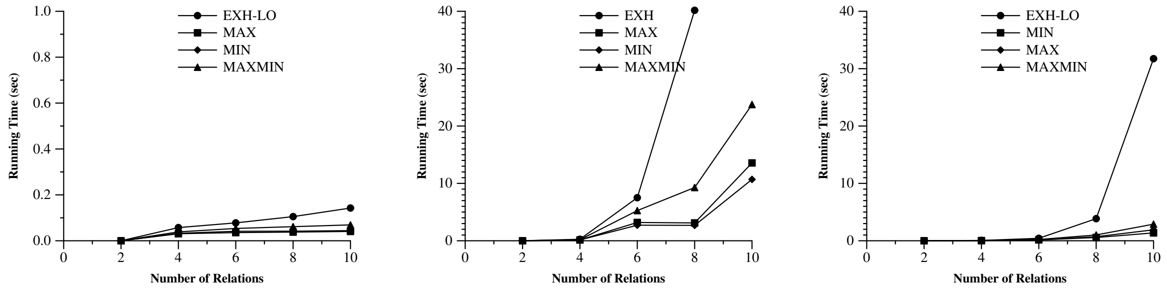
Considering that the two greedy heuristics perform better under different scenarios, we experimented with a hybrid version of the two called MINMAX. This heuristic simply runs both the heuristics simultaneously and chooses the one with better cost. As we can see, this heuristic performs much better than either MIN or MAX, though still worse than the exhaustive algorithm.

6.4.2 Running Time of the Algorithms

Figure 20 shows the running times of the estimation planning algorithms. As we can see, the running time of the EXH-LO algorithm makes it impractical in pretty much all scenarios. As expected, the greedy heuristics MAX and MIN take much less time. The MINMAX heuristic also finds an estimation plan very quickly. Note that, since the MIN and MAX algorithms share a lot of computation (more precisely, creation of some skeleton histograms), the time of MINMAX is usually much less that the combined time for MIN and MAX.

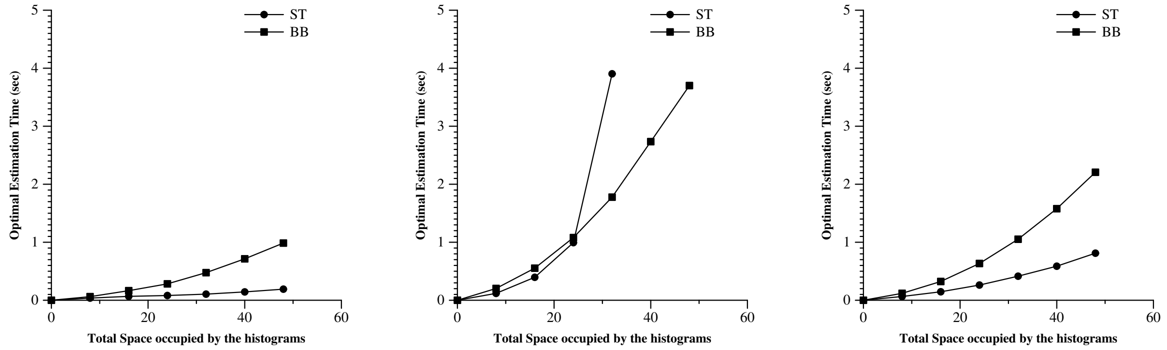
6.5 Which representation to use ?

As we have seen, the choice of the algorithms to use and their performance is greatly affected by the representation used to store the intermediate results. This begs the question : which representation should be used during the optimization process ? BB-representation has the advantage that it is very simple to use and implement, and also does not lead to the same kind of variability as seen



Path Query | Star Query | Two Stars Query

Figure 20: Running Times of the algorithms (ST Representation)



Path Query (10 Relations) | Star Query (8 Relations) | Two Stars Query (8 Relations)

Figure 21: Comparison between the two representations

with the ST-representation. On the other hand, if the order in which attributes are collapsed is optimal, the ST-representation can result in huge savings in estimation time. This is mainly because of two factors : (1) the split-tree structure can guide the join procedure very effectively to eliminate unnecessary comparisons between buckets, and (2) the projection operation can result in small histograms because this representation eliminates any overlap between buckets within a histogram.

Figure 21 shows the results of an experiment conducted to show the difference between estimation costs based on the representation used. As with the first experiment (Section 6.2, the histogram sizes were chosen to be equal and the total space occupied by the histograms was varied between 8KB and 48KB. As we can see, for the two-stars query and the path query, the ST representation results in significant savings in estimation cost, whereas for a star query on 8 relations, the BB-representation performs better. In fact, if the ST-representation is used for the star query, the intermediate histogram sizes are so large that it is not possible to execute the estimation plan for large histogram sizes. This is mainly because of the 7-dimensional input histogram that, in general, produces very large projections.

This experiment suggests that the ST-representation should be used in general, except when large high-dimensionality histograms are present in the input. A hybrid algorithm that switches between the two representations is not feasible as it is computationally very expensive to convert from the BB-representation to the ST-representation.

7 Conclusions

With the increasing use of complex summarization techniques for better cardinality estimation, the question of using these optimally becomes very important as the computational cost of using these complex techniques can be quite high. In this paper, we show how this problem naturally arises when a System R-style optimization algorithm is extended to use multi-dimensional histograms, and we develop algorithms for finding good estimation plans to compute the required estimates. Our experimental results validate both the need for better estimation planning and the performance of our algorithms.

We hope to extend this work in many directions. One of the important questions that needs to be answered is whether the complexity of the estimation process can be reduced significantly if the intermediate synopses that are constructed are not required to be as accurate as possible given the base table synopses. We also plan to address different query optimization algorithms, as the intermediate relations that are encountered during the process depend significantly on the optimization algorithm used. For example, randomized algorithms [IK90] require costing only a limited number of query plans, and the estimation procedure should be able to estimate only those intermediate table cardinalities that are required efficiently. Approximate query answering is another area where the techniques developed in this paper could be used. With increasing use of complex synopsis techniques for this purpose, approximate querying can be computationally expensive for large queries, and as such, estimation planning may have to be employed for efficient use of the synopses.

References

[AC99] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning Histograms: Building Histograms Without Looking at Data. In *SIGMOD*, 1999.

[Ant93] Gennady Antoshenkov. Dynamic query optimization in rdb/vms. In *ICDE*, 1993.

[BCG01] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. STHoles: A Multidimensional Workload-aware Histogram. In *SIGMOD*, 2001.

[BDF+97] Daniel Barbará, William DuMouchel, Christos Faloutsos, Peter J. Haas, Joseph M. Hellerstein, Yannis E. Ioannidis, H. V. Jagadish, Theodore Johnson, Raymond T. Ng, Viswanath Poosala, Kenneth A. Ross, and Kenneth C. Sevcik. The new jersey data reduction report. *IEEE Data Engineering Bulletin*, 20(4), 1997.

[CGRS00] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. In *VLDB*, 2000.

[Chr83] Stavros Christodoulakis. Estimating block transfers and join sizes. In David J. DeWitt and Georges Gardarin, editors, *SIGMOD*, 1983.

[Col00] Richard Cole. A decision theoretic cost model for dynamic plans. *IEEE Data Engineering Bulletin*, 2000.

[DGR01] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *SIGMOD*, 2001.

[GGgz] Minos Garofalakis and Phillip Gibbons. Tutorial on "Approximate Query Processing: Taming the Terabytes". In *VLDB, 2001*, http://www.bell-labs.com/user/minos/Talks/vldb01-tutorial.ppt.gz.

[Gra95] Goetz Graefe. The cascades framework for query optimization. *IEEE Data Engineering Bulletin*, 1995.

[GTK01] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, 2001.

[IK90] Yannis E. Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD*, 1990.

[KBZ86] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In *VLDB*, 1986.

[MCS88] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3), 1988.

[MD88] M. Muralikrishna and David J. DeWitt. Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries. In *SIGMOD*, 1988.

[MVW00] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *VLDB*, 2000.

[OL90] Kiyoshi Ono and Guy M. Lohman. Measuring the complexity of join enumeration in query optimization. In *VLDB*, 1990.

[PI97] Viswanath Poosala and Yannis E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *VLDB*, 1997.

[PSC84] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD*, 1984.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD*, 1979.

[SBM93] K. Seppi, J. Barnes, and C. Morris. A bayesian approach to database query optimization. *ORSA J. Comp.*, 1993.

[VWI98] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. Data Cube Approximation and Histograms via Wavelets. In *CIKM*, 1998.

[Wil91] Dan E. Willard. Optimal sample cost residues for differential database batch query problems. *JACM*, 38(1), 1991.

A Project and join operations on ST-representation MHists

A.1 Projection

Let's say we want to project a d-dimensional MHist on attributes $A_1, \ldots, A_d$ on attributes $A_1, \ldots, A_{d-1}$. Consider the simple case where only the top split (the very first split made during the construction) is on attribute A_d, whereas rest of the splits on the other dimensions. Let T_l and T_r be the two children of the root node. Note that both these split-trees represent a partition of the same $(d-1)$-dimensional space. If we want to project A_d out, then these this particular $(d-1)$-dimensional space must be split so that it agrees with both T_r and T_l. This is done by iterating over all buckets of T_l and splitting the space occupied by each of the buckets according to T_r. This is done by first finding out which of the splits contained in T_r are relevant to this bucket and then appending that portion of T_r to the corresponding leaf. Note that, this particular operation can also be thought of as a *spatial join* between the buckets of T_l and T_r. We do not use a spatial join algorithm for this, but such an algorithm could be used if the naive algorithm turns out to be computationally expensive.

The projection algorithm works bottom-up using the above idea to eliminate any splits on A_d recursively. New bucket frequencies are computed during this process as well. Please refer to [DGR01] for details. Projections that involve projecting out multiple attributes can be done simultaneously using this algorithm.

A.2 Multiplication

Multiplication uses a very similar algorithm as above. As a matter fact, both these operations use a common routine as described in [DGR01]. Let us say we want to multiply two histograms H_1 and H_2 that are on attributes $A_1, \ldots, A_d, B_1, \ldots, B_e$ and $A_1, \ldots, A_d, C_1, \ldots, C_f$ resp., to get a histogram on attributes $A_1, \ldots, A_d, B_1, \ldots, B_e, C_1, \ldots, C_f$. This operation can be thought of as an spatial join where two buckets are combined to produce a new bucket in the result if there is an overlap in the ranges of attributes $A_1, \ldots, A_d$. We implement this by iterating over all the buckets of H_1 and finding the relevant splits in H_2 that split the space occupied by that bucket. Again, more sophisticated spatial join algorithms could be used instead, but the complexity of the above operation is upper bounded by the $|H_1||H_2|$, where $|H|$ denotes the number of buckets in these histograms, and this number is never very large as long as the intermediate histograms are created intelligently.

B Proofs

Here, we will briefly outline proofs of the claims we make in this paper.

Theorem B.1 *The number of ways of computing all the intermediate relation cardinalities for a n-relation star query is larger than $O(2^{2^n})$, whereas for a path query, there are at least $O(n^{n^2})$ distinct possibilities.*

Proof: Both these derivations use the fact that the cardinality of an intermediate relation may be estimated using histograms on any two intermediate relations that can generate this relation through a join, and the choice made for any intermediate relation is independent of the choice for any other intermediate relation.
Star query: Given an intermediate relation consisting of k base relations, there are $k-1$ possible joins that can generate this relation. Since there are $\binom{n-1}{k-1}$ intermediate relations with k base relations each, this gives us the following formula for the total number of possibilities :

$$\Pi_{k=2}^{n-1}(k-1)^{\binom{n-1}{k-1}} \gg 2^{\Sigma\binom{n-1}{k-1}} = 2^{2^{n-1}}$$

Note that this is a very conservative estimate.
Path query : There are $k-1$ different joins that can generate any given intermediate relation consisting of k base relations. This gives the following expression for the total number of possibilities :

$$\Pi_{k=1}^{n-1}(n-k+1)^{k-1} \gg \Pi_{k=1}^{n/2}(n-k+1)^{k-1} \gg (\frac{n}{2})^{O(n^2)}$$

∎

B.1 Proof of Theorem 4.2

We will prove this by constructing a correspondence between intermediate relations encountered during the optimization algorithm and the subproblems generated during our search algorithm.

The first subproblem, which is also the original problem, is assigned to the query result, *ie.*, the intermediate relation that consists of all base relations. Let (R, S) be a leaf edge in this problem, and let R be the relation with degree 1. Collapsing this edge results in exactly one subproblem for our search algorithm, though during estimation, this subproblem needs to be executed with two different sets of input histograms. We will assign this subproblem to the intermediate relation that consists of all base relations except the relation R. As we can see, each subproblem that can be generated at this stage will be assigned to a different intermediate relation through this process. By proceeding recursively on the subproblems, a correspondence can be constructed that assigns each subproblem to a unique intermediate relation.

∎