Multi-criteria Query Optimization in the Presence of Result Size and Quality Tradeoffs

Lakshmi Priya Mahalingam (priyam@asu.edu) and K. Selçuk Candan (candan@asu.edu)
Computer Science and Eng. Dept., Arizona State Univ., Tempe, AZ 85283, USA.

Abstract. In this paper, we present novel multi-criteria query optimization techniques for performing query optimization in databases, such as multimedia and web databases, which rely on imperfect access mechanisms and top-k predicates. We present an optimization model that (1) takes into account different binding patterns associated with query predicates, (2) considers the variations in the expected query result sizes as a function of query execution plans, and (3) considers the expected result qualities of the execution orders. We address the complexity and the well-known NP-complete nature of the query optimization problem by *adaptively* reducing the granularity of the search space. For this purpose, unlike the data histograms which capture the data distribution, we propose opt-histograms that capture the distribution of sub-query-plan values over many optimization tasks.

1. Introduction

Multimedia database queries have characteristics different from queries in traditional databases. For instance, in traditional, such as relational, databases the number of results of a query is independent of the query execution order. In multimedia and web databases, however, due to the use of thresholds and top-k predicates [1, 2] the number of query results can change depending on the query execution order. Furthermore, finding exact matches is not required (impossible in many cases) in multimedia and web databases; results are ordered according to their degrees of match to the query. Due to these inherent differences, we need novel query processing and optimization algorithms. For example, let us consider an SQL-like multimedia query:

```
select location, time, image
from surveillance_scans(location, time, image)
where location = "entrance_gate" and
        extract_pattern(image, pattern) and
        match_pattern(pattern,"alert.gif").
```

In this application, surveillance images are continuously fed from the cameras into a database to identify an alert condition; i.e., we would like to identify the *best alert candidates* in the *shortest amount of time*. While optimizing such a query, we need to consider the following issues:

Overloaded implementations of query predicates. Media-related predicates can be implemented by various user-defined functions or indexes corresponding to different ways the predicate can be invoked.

For instance, the query predicate, `extract_pattern(image,pattern)`, can have three different overloaded implementations:

- given an image, one implementation extracts a predetermined number of patterns using a pattern recognition function,
- given a pair of an image and a pattern, another one searches for the pattern in the image using a neural-net engine, and
- given a pattern, the last one retrieves all matching images using a cache of pre-extracted of pattern/image pairs maintained in a multi-dimensional index structure.

Thresholds and "top-k" predicates. In the above example, every given pattern matches every other one to some degree. Therefore, providing all answers is neither feasible nor desirable. Consequently, multimedia functions are usually implemented as *thresholds* or *top-k retrieval* functions. This means that a sub-query or a user defined function returns only a small, but most relevant, portion of all possible results. However, if two overloaded implementations of the same predicate define *relevance* differently, then they can return different portions of the result space. One resulting complication is that a query can return different numbers of tuples depending on the execution plan chosen by the query optimizer:

EXAMPLE 1.1. Let us assume that we can execute the query described earlier in two different ways:

- **(a1)** Find all surveillance images scanned at the "entrance gate"; **(a2)** find the patterns in each image; and **(a3)** return those images that contain a pattern which match "alert.gif".
- **(b1)** Find all patterns that resemble "alert.gif" using an index structure; **(b2)** find all images which contain one of these patterns; and **(b3)** return those that are scanned at the "entrance gate".

There is no reason to expect that these two executions will return the same number of tuples. At the second step of the process, different overloaded implementations (or binding patterns) of `extract_pattern (Image, Pattern)` may use different retrieval policies: in **(a2)** the result space may be pruned to the top-5 patterns in the image, whereas in **(b2)** all images containing the given pattern may be returned. □

Approximate matches and ranking. The expected result qualities of predicates may also vary depending on how a predicate is executed. For instance, in the above example,

- if the predicate `extract_pattern (image, pattern)` uses a multi-dimensional index structure to find all images that contain a pattern, the quality of the results will depend on the precision and recall of the index structure; whereas

– if it extracts all patterns from a given image using a precise image processing engine, then although the process may be expensive, the results may be of high quality.

In general, it may be possible to *estimate* the expected quality associated with a particular retrieval strategy by maintaining appropriate statistics. In Section 3 we discuss the semantics behind this concept.

1.1. Summary

In most cases, there is a trade-off between query execution cost, the number of tuples returned, and the expected quality of the results. We can highlight the contributions of this paper as follows:

– Unlike previous work which deals with restrictive-expensive predicates [3] (selections), we consider query optimization with arbitrary non-restrictive predicates.
– We introduce the size (fanout) and the quality factors (that vary with the binding patterns and execution plans) into the cost model and optimization algorithms.
– We reduce the granularity of the search space to reduce the complexity of the optimization task. We propose a novel histogram based approximation technique, where opt-histograms describe the nature of previously conducted optimization tasks.

In the next section, we first discuss the impact of top-k predicates in query optimization. Then, in the following section, we extend the framework to handle varying expected qualities of the results.

2. Optimization in the Presence of Top-k Predicates

Multimedia queries are generally processed in two ways: **(1)** using regular query processing techniques within database management systems and **(2)** using special merge techniques that benefit from the fact that results returned by most multimedia user-defined functions are ranked based on their scores [15, 4, 16, 2, 5].

Merge-based query processing for top-k retrieval does not lend itself to explicit query optimization; i.e., the merge strategy is predetermined; hence there usually are no query processing alternatives. Although these algorithms minimize the number of merged tuples, they do not necessarily optimize the cost of processing the non-fuzzy parts of the query. Furthermore, they do not address the fact that even the fuzzy predicates can have different behaviors based on the binding pattern utilized for invoking them. Therefore, in this paper, we develop a model which can lend itself to query optimization in the traditional sense instead of applying a merge strategy.

A query optimizer needs an accurate cost estimation technique. It also has to use an efficient search space enumeration algorithm to identify the relevant parts of the search space. An optimization model assigns a value (to be minimized) to all partial or complete plans in the search space. It also determines the output size of the data stream for every operator and predicate in the plan. This plan must respect the available binding patterns of each query predicate. A binding pattern of a predicate specifies which attributes of a relation must be bound to access it. For example, a predicate $r(x, y)$, with two parameters can have four binding patterns:

- ff (or, alternatively, $\emptyset$) when both parameters are free,
- bf (or, alternatively, $\{x\}$) when x is bound and y is free,
- fb (or, alternatively, $\{y\}$) when x is free and y is bound, and
- bb (or, alternatively, $\{x, y\}$) when both parameters are bound.

Each binding pattern abstracts a different implementation of the predicate in terms of an external function or an index structure. Therefore, given a multimedia query predicate, each binding pattern of predicate has different *cost* and *size* parameters associated with it. This research aims at finding an optimal query execution plan, given a set of predicates, and the associated binding patterns, each with different execution cost and size characteristics.

2.1. Query Execution Cost and Result Size (Fanout)

In this subsection, we introduce two parameters, *cost* and $fanout$, involved in the optimization model we propose, instead of the *table scan cost* and *selectivity* parameters traditionally used for optimizing queries. The *cost* and $fanout$ parameters are defined *per binding pattern*. For example, given a predicate $r_1(x, y)$, $cost^{bf}(r_1(x, y))$ denotes the expected cost of accessing r_1 by binding the parameter x to a constant and retrieving all matching values for y. Similarly, $fanout^{bf}(r_1(x, y))$ denotes the expected number of $\langle x, y\rangle$ when r_1 is accessed by binding the parameter x to a constant.

2.1.1. *The "cost" Parameter*

The *cost* parameter is defined as a function of the number of inputs to the bound parameters of the predicate. Note that this definition is similar to the cost of a limited access pattern as used in [6]. If a predicate, say $r_1(x, y)$ is executed first during the processing of a join query $r_1(x, y) \bowtie r_2(y, z)$, then there will be a set, $\mathcal{Y}$, of y values that are to be passed to r_2. In a pipelined implementation, each y value will be processed separately by r_2 to identify the corresponding z

values. The $cost^{bf}(r_2(y,z))$ parameter, in this case, simply measures the expected cost of r_2 for each input value of y. In some cases, operating r_2 on individual input tuples may have too much overhead. In such a case, the value of the $cost^{bf}(r_2(y,z))$ parameter denotes the cost of the *batch* execution of r_2 averaged over the number of inputs.

2.1.2. *The "fanout" Parameter*

The $fanout$ parameter denotes the expected number of outputs as a function of the number of inputs to the bound parameters of the predicate. For example, $fanout^{bf}(r_1(x,y))$ denotes the expected number of $\langle x,y\rangle$ pairs that the predicate returns per each bound value of x. Consequently, the fanout parameter can be larger than 1.0. This definition is inherently different from the definition of *selectivities* used in query optimization literature [3, 7]:

- a top-k retrieval predicate will have a predetermined number of results (k) independent of the value of x or the size of the actual result space. For example, `extract_pattern(inputimage, outputpattern)` described in the Introduction extracts at most 5 patterns per input image, independent of the complexity of the image.
- since we are not necessarily referring to a table with a known number of tuples, the traditional definition of selectivity does not apply. For example the predicate `extract_pattern(inputimage, outputpattern)` extracts whatever patterns it can find in a given image. Different algorithms would be able to identify a different number of patterns in the same image.

Whenever they are already available, it is possible to map selectivities to fanouts. For example, if we are given a table with schema $t(x,y)$ which has $num(t(x,y))$ rows and selectivity $sel^{bf}(t(x,y))$, then we can calculate the corresponding fanout as

$$fanout^{bf}(t(x,y)) = sel^{bf}(t(x,y)) \times num(t(x,y)).$$

The reverse of this, however, is not true; that is, given a fanout, it is not always possible to map it into a selectivity.

2.2. Modeling the Desirability of a Query Plan

The above semantics of predicate cost and fanout necessitate a corresponding model to evaluate the *desirability*[1] of a query plan. Given

- **(predicates)** a set of predicates $\mathcal{P} = p_1, \ldots p_n$,

[1] Here, we use the term desirability instead of optimality as, in the literature, optimality is generally used to mean minimal cost.

- **(attributes)** a set of attributes $\mathcal{A} = a_1, \ldots, a_m$,
- **(query)** a join query $q(a_1, \ldots, a_m) = p_1(a_1, \ldots, a_m) \bowtie \ldots \bowtie p_n(a_1, \ldots, a_m)$,
- **(inputs)** an input binding pattern, $in\_bound \subseteq \mathcal{A}$, which denotes the set of attributes that are bound in the beginning. The set, in_free, of attributes that are free in the input can be computed as $in\_free = \mathcal{A} - in\_bound$,
- **(outputs and intermediaries)** an output binding pattern, $out\_bound \subseteq \mathcal{A}$, which denotes the set of attributes that are expected to be bound at the end of the process. The set, in_bound, is a proper subset of the set out_bound and the set, out_free, of attributes that are free in the output can be computed as $out\_free = \mathcal{A} - out\_bound$,
- **(available predicate implementations)** for each predicate $p_i \in \mathcal{P}$, a binding function
  $bind_i : 2^{\mathcal{A}} \rightarrow \{2^{\mathcal{A}} \cup \bot\}$, such that given an input binding pattern, in, we have $bind_i(in) \supset in$ if the input binding pattern is allowed (i.e., there is an implementation of this predicate for this input pattern) and $bind_i(in) = \bot$ if it is not

which describes the acceptable input/output binding relationships, we can define the cost and the fanout of the query as

$$\begin{array}{l}
\langle\ cost^{out_bound}(q)\ ,\ fanout^{out_bound}(q)\ \rangle\ = \\
\quad \textbf{best_of}\ \ \{\ \langle cost, fanout\rangle \quad \textbf{such that} \\
\qquad\qquad cost = join_cost(\langle cost^{pat_1}(p_1), fanout^{pat_1}(p_1)\rangle, \ldots, \\
\qquad\qquad\qquad\qquad \langle cost^{pat_n}(p_n), fanout^{pat_n}(p_n)\rangle)\ \wedge \\
\qquad\qquad fanout = join_fanout(fanout^{pat_1}(p_1) \ldots, fanout^{pat_n}(p_n)) \quad \wedge \\
\qquad\qquad (\mathcal{A} - pat_1) \cup (\mathcal{A} - pat_2) \cup \ldots \cup (\mathcal{A} - pat_n) = in_free \quad \wedge \\
\qquad\qquad bind_1(pat_1) \cup bind_2(pat_2) \cup \ldots \cup bind_n(pat_n) = out_bound \\
\quad \},
\end{array}$$

where

- pat_i is an input binding pattern for predicate p_i,
- $join\_cost$ is the combined cost of the join, and
- $join\_fanout$ is the combined fanout value.

The last two conditions ensure that the input/output binding constraints are observed:

- the union of the attributes that were free when each predicate is called must be equal to the set of attributes that were free at the very beginning, and
- the union of attributes that were bound as a result of the evaluation of each predicate must be equal to the set of attributes that are expected to be bound at the end.

Computations of $join\_cost$ and $join\_fanout$ depend on the semantics and the actual implementation of the join operators. The following example assumes a nested-loop join implementation.

EXAMPLE 2.1. Using a nested loop join operator, the best way of computing $q(x, y, z) = r_1(x, y) \bowtie r_2(y, z)$ can be found by selecting the *best* among

$$\begin{aligned}
\langle cost^{bbb}(q(x,y,z)), fanout^{bbb}(q(x,y,z))\rangle = \quad & \textbf{best_of \{} \\
& \langle cost^{ff}(r_1(x,y)) + fanout^{ff}(r_1(x,y)) \times cost^{bf}(r_2(y,z)), \\
& \qquad fanout^{ff}(r_1(x,y)) \times fanout^{bf}(r_2(y,z))\rangle, \\
& \langle cost^{ff}(r_2(y,z)) + fanout^{ff}(r_2(y,z)) \times cost^{fb}(r_1(x,y)), \\
& \qquad fanout^{ff}(r_2(y,z)) \times fanout^{fb}(r_1(x,y))\rangle, \\
& \langle cost^{ff}(r_1(x,y)) + cost^{ff}(r_2(y,z)) + fanout^{ff}(r_1(x,y)) \times fanout^{ff}(r_2(y,z)), \\
& \qquad fanout^{ff}(r_1(x,y)) \times fanout^{ff}(r_2(y,z))\rangle.\textbf{\}}
\end{aligned}$$

This formulation associates an execution cost ($cost^{bbb}$) and result size ($fanout^{bbb}$) for each option:

- In the first case, x and y have been instantiated by calling r_1 with free variables, and then these values have been validated by accessing r_2 with y bound.
- The second case is similar to the first case, except that the orders of r_1 and r_2 are exchanged.
- In the third case, both relations are independently accessed using both attributes free, and the results have been joined together.

The result size, in each case, is the product of the expected result sizes associated with the corresponding binding patterns. □

In traditional models, it is relatively straightforward to choose the **best** plan among the alternative query processing plans. In the above example, traditional cost models would recognize that all possible executions would result in the *same* number of tuples. Therefore, irrespective of the plan chosen, the final size for the above example would be the same. Hence, we can choose the *best* plan by identifying the plan with the smallest *cost*. The proposed model, on the other hand, differs from the traditional models in that it has to account for different result sizes for different execution orders.

2.3. Alternative Ways to Define the Desirability of a Query Plan

We see three main alternative ways to define desirability:

- **min_cost:** When the goal is to process the query as quickly as possible (irrespective of the number of resulting tuples), we can define best_of as min_cost. In this model, given a set of $\langle cost, fanout\rangle$ pairs, we would choose the pair with the smallest cost.

- **min_unit_cost :** When we are aiming to generate as many results as possible with the smallest cost, we can define the best_of as min_unit_cost, where unit_cost is defined as $\frac{cost}{fanout}$. When the number of results may change based on the way query is processed, unit_cost may be a better performance indicator.
- **min_fanout:** When the user wants to find a plan which returns the minimum number of results, we can define best_of as the plan with the smallest fanout. Note that, min_fanout is especially important when there is a significant penalty/cost associated with accessing individual results.

2.4. Query Optimization Algorithms based on Desirability

Most cost-based query optimization algorithms rely on dynamic programming [7], which builds query plans, in stages, in a bottom-up and breadth first manner. The algorithm considers subgoals with increasing number of joins in them. Since, through the use of a *table* that records intermediary results, the recomputation of results are avoided whenever sub-goals overlap, the execution time is greatly reduced[2].

One desirable property of a *unit_cost*-based model is that, under certain conditions, optimal (in terms of unit cost) plans have optimal (in terms of unit cost) subplans. Therefore, we can divide the optimization problem into smaller problems and solve each one of them recursively. Furthermore, we can rely on a *table* to keep subquery results to avoid re-evaluation of overlapping subqueries. Therefore, when the **best_of** measure is defined as **min_unit_cost** we can benefit from dynamic programming. Note that the **min_fanout**-based model, under certain conditions, may also lend itself to dynamic programming based optimization.

When the **min_cost** based optimization is required, however, we can not benefit from these algorithms. One major disadvantage of this formulation is that optimal (in terms of cost) plans may not have optimal (in terms of cost) subplans. This is due to the fact that logically equivalent sub-plans may have different fanouts, and when the cost parameter is used to select among alternative subplans, we may not reach an optimal overall plan. Consequently, we can not use any recursively structured algorithm, such as dynamic programming. In this case, one solution is to use the **min_unit_cost** as a heuristic to reduce the search space at every level of a dynamic programming algorithm. In other words, given a dynamic-programming based optimization algorithm,

[2] Since dynamic programming-based query optimization algorithms [7] are common and well-understood, we do not provide the pseudo-code of such an algorithm in this paper

at each level, we can **(1)** rank sub-plans based on their *unit-cost*s, **(2)** prune-away sub-plans with large *unit-cost*s, and **(3)** consider only those plans with small *unit-costs*. The amount of pruning can be controlled to achieve different levels of optimization speed and optimality.

Note that the concept of *unit-cost* is different from the concept of *rank* [8, 3] used for dealing with expensive predicates. A single *rank* is associated with every predicate in a query and these predicate ranks are used as heuristics to improve query plans. In the model we present, however, *unit-cost*s are associated with alternative sub-plans, not with predicates themselves. Furthermore, *unit-cost*s are functions of the binding patterns of the predicates , whereas in [8, 3], each predicate has a single rank.

3. Optimization in the Presence of Fuzziness

It is possible to classify the fuzziness in the multimedia queries into two main categories: *precision-related* and *recall-related*. These capture fuzziness due to similarity of features, imperfections in the feature extraction algorithms, imperfections in the query formulation methods, and the precision and recall rate of the utilized clustering algorithms and index structures.

EXAMPLE 3.1. Let us assume that we are given a query of the form
$Q(X) \leftarrow s\_like(man, X.semantic\_property) \wedge$
$image\_match(X.image\_property, \text{“}a.gif\text{”})$.
For a given object I, let the corresponding $semantic\_property$ be $woman$ and $image\_property$ be a $im.gif$. Let us assume that the semantic precision of s_like is 0.8 and the image matching precision $image\_match$ is 0.6. This means that the index structure and semantic clusters used to implement the predicate s_like guarantee that 80% of the returned results are semantically similar to a given string. Similarly, the predicate $image\_match$ guarantees that 60% of the returned results are visually similar to a given image. Then, assuming that the two predicates are not correlated, $Q(I)$ should be $0.8 \times 0.6 = 0.48$: By replacing X in $s\_like(man, X)$ with $woman$, we maintain 80% precision; then, by replacing the X in $image\_match(X, \text{“}a.gif\text{”})$ with I, we maintain 60% of the remaining precision. The final precision (or confidence) is 0.48. □

Note that in information retrieval, the precision/recall values are mainly used in evaluating the effectiveness of a given retrieval operation or the effectiveness of a given index structure. We, on the other hand, are using these terms more as statistics which can be utilized to estimate the quality of a given query plan.

The above example shows that *product semantics* can be a useful tool for merging the expected qualities of predicates in a given query

plan. Other merge functions, such as *min* and *arithmetic average* have been studied in literature in the context of fuzzy relational databases and probabilistic databases [9]. The focus in these works, however, has been on associating a score value for each resulting tuple, instead of associating an expected quality with each query plan. Therefore, we next provide an overview of the plan qualities.

3.1. Extending the Model with Expected Qualities

Based on these observations, we extend the proposed model with expected qualities of alternative query execution plans. In other words, given a join query $q(a_1, \ldots, a_m) = p_1(a_1, \ldots, a_m) \bowtie \ldots \bowtie p_n(a_1, \ldots, a_m)$, we define the cost, the fanout, and the quality of the result as

$$
\begin{array}{l}
\langle\ cost^{out\_bound}(q)\ ,\ fanout^{out\_bound}(q)\ ,\ quality^{out\_bound}(q)\ \rangle\ = \\
\quad \textbf{best\_of}\ \ \{\ \langle cost, fanout, quality\rangle\ \textbf{such that} \\
\qquad cost = join\_cost(\langle cost^{pat_1}(p_1), fanout^{pat_1}(p_1)\rangle, \ldots, \\
\qquad\qquad \langle cost^{pat_n}(p_n), fanout^{pat_n}(p_n)\rangle)\ \wedge \\
\qquad fanout = join\_fanout(fanout^{pat_1}(p_1) \ldots, fanout^{pat_n}(p_n))\ \ \wedge \\
\qquad quality = join\_quality(quality^{pat_1}(p_1), \ldots, quality^{pat_n}(p_n))\ \ \wedge \\
\qquad (\mathcal{A} - pat_1) \cup (\mathcal{A} - pat_2) \cup \ldots \cup (\mathcal{A} - pat_n) = in\_free\ \ \wedge \\
\qquad bind_1(pat_1) \cup bind_2(pat_2) \cup \ldots \cup bind_n(pat_n) = out\_bound \\
\quad \},
\end{array}
$$

where *join_quality* is the combined quality (depending on the join operator used) as a function of the input qualities. If we would like to use the *join_quality* along with the cost in a standard dynamic-programming (System-R style) query optimizer [7], we need to ensure the optimality property of the subqueries. In that case, we need to use a merge function, such as *min* or *product*, which can provide this property. We have also seen in earlier examples that the product semantics is a good choice for merging the expected qualities.

EXAMPLE 3.2. If we reconsider the query we presented in Example 2.1, i.e., $q(x, y, z) = r_1(x, y) \bowtie r_2(y, z)$, and if we assume *product* to merge the qualities of the fuzzy predicates, then

$$
\begin{array}{lll}
quality^{bbb}(q) & \in & \{\ quality^{ff}(r_1(x,y)) * quality^{bf}(r_2(y,z)), \\
 & & \quad quality^{ff}(r_2(y,z)) * quality^{fb}(r_1(x,y)), \\
 & & \quad quality^{ff}(r_1(x,y)) * quality^{ff}(r_2(y,z))\}.
\end{array}
$$

The product model used in this example is based on the observation that each subsequent operation reduces (and never increases) the expected quality of the final result. □

3.2. Merging Cost and Quality

In order to account for the quality information in query optimization, we need to extend the **best-of** definition to include quality. Assuming

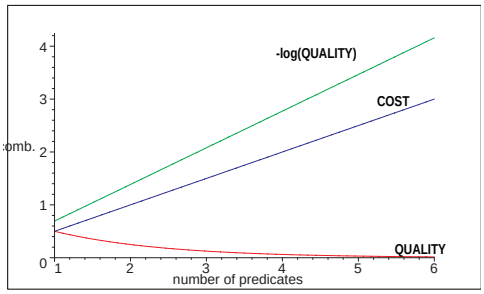

Figure 1. The general behaviors of *quality* and *cost* parameters; $-log(quality)$ behaves like the *cost* parameter. This figure assumes that the both cost and quality of each binding pattern of each predicate are 0.5. Different values gives different curves, but the overall behavior is the same.

that the user specifies a trade-off between cost and quality using two weights, w_1 and w_2, we need to merge cost and quality of each sub-plan to calculate its *desirability*. Note that cost increases additively for each predicate involved, quality decreases multiplicatively (Figure 1). Therefore, the **min_cost** semantics of the **best_of** function, can be extended as $w_1 \times cost + w_2 \times log(qual)$, to merge cost and quality. The other **best-of** semantics, **min_unit_cost** and **min_fanout**, can also be extended in a similar manner.

4. Adaptive Reduction of Search Space Granularity

Relational query optimization is an NP-complete problem. Therefore, in addition to pruning the search space as described in Section 2, we propose to reduce the granularity of the search space to further bring down the complexity of query execution.

We divide the search space into buckets, whose sizes are determined by a given granularity. All sub-plans that belong to a particular interval of values are placed in the same bucket. In each step of query-optimization, instead of using all sub-plans, only one sub-plan from every bucket is used. The number of sub-plans to be considered can be significantly reduced by approximating the costs of the sub-plans and choosing only the best of them. The desired level of approximation needed is described as the granularity of the distribution. If the granularity is small, there potentially is a considerable reduction in optimization time, but this introduces errors.

In general, identifying the boundaries and widths of the buckets is not trivial. In this section, we introduce *opt-histograms* that can be used for intelligently choosing the granularity (places of bucket boundaries) of the search space. Opt-histograms describe the optimization statistics accumulated by observing similar queries optimized over time.

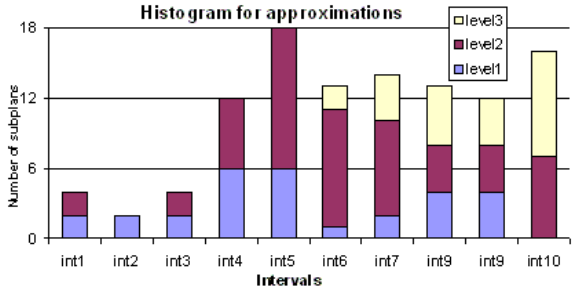

Figure 2. An example opt-histogram, which shows that the relevant space of costs varies with the query optimization stages

One, relatively strict, way to define similar queries is to define them as queries of the same type; in other words, two queries are of the same type if they are equal modulo input parameter values. It is possible to relax the definition of the similarity to those queries with the same structure but alternative media-predicates or search range (for instance, the MPEG7 standard defines various descriptors to describe the content of an image or an object of interest within an image). Once we identify similar queries, we can use opt-histograms to intelligently choose the best way to modify the granularity of the search space.

An opt-histogram is generated by analyzing the query optimization history. Figure 2 shows an example opt-histogram generated using a set of similar queries. The opt-histogram shows the distribution of the estimated costs of sub-plans that belong to different levels of join ordering. This histogram was generated over 1000 optimization tasks using a 4-relation query. The levels in the figure denote the 2-, 3-, 4- way join stages of the query optimization process. As shown in this figure, we observed a consistent behavior of sub-plan execution costs in different query optimization stages. Relevant regions of the cost-space shifts at different stages of the query optimization process. Therefore, different boundaries should be used at different stages of query optimization. One property of the opt-histograms that is special to the multimedia and web databases is that sizes show different characteristics than the costs. Sizes can be defined multiplicatively (unlike the additively defined cost). Therefore, they require non-uniform granularity reductions of the space. In this paper, we do not discuss *size histograms*.

In Section 2, we showed that when we are dealing with the **best_cost** metric, we can implement a dynamic-programming based algorithm which selects the best k ranked sub-plans for every two-way join. Given an opt-histogram, we can fursther reduce the search space, by coalescing the sub-plans using a low granularity search space. All plans that fall into a particular bucket are approximated as being equivalent and this

reduces the search space further. Next, we experimentally evaluate the effectiveness of the proposed solutions.

5. Experiments

We first describe the setup of the experiment environment. In order to observe the behavior of the proposed query optimization algorithm under very different conditions, we opted to use synthetically generated databases and queries. The experiments were conducted with queries with 4 joining relations each with 2-3 attributes (eg., $R1(x, y) \bowtie R2(y, z, w) \bowtie R3(x, w) \bowtie R4(z, w, t)$). The number of join attributes also varied randomly from query to query. In order to observe the worst-case optimization cost, we allowed all possible binding patterns. In different runs, each predicate got associated with randomly generated data that follow random *cost* and *size* patterns. These are generated keeping in mind the characteristics of multimedia and web predicates. We varied the expected cost, size, and quality values of the binding patterns of each predicate across different runs to observe the performance of the proposed optimization algorithm under different input parameters. The cost values were randomly generated, for every binding pattern, between 1 and 1000. The size and quality parameters were randomly generated between 0.0 and 1.0. We have experimented with both uniform and biased distributions (for example, small number of very expensive binding patterns); the results were comparable. In order to compare the performances (in terms of time as well as the closeness to optimality) of various approaches, we have implemented the following:

- *Optimal:* This algorithm enumerates all plans with no pruning or approximations.
- *Predicate migration:* We have also implemented the predicate migration algorithm [3]. Note that predicate migration uses only one rank per predicate. It is essentially a heuristic when applied to the model presented in this paper. Therefore, to evaluate its performance conservatively, for each predicate we used the worst rank among all its binding patterns.
- *1-Optimal:* This algorithm prunes the search space choosing only one best plan for each sub-goal.
- *K-Optimal:* This algorithm keeps k best plans for each sub-goal (3 in these experiments).
- *K-Hist:* This algorithm approximates the top k ranked sub-plans using a reduced granularity search space.

The following sections discuss how these algorithms perform with respect to cost and size factors.

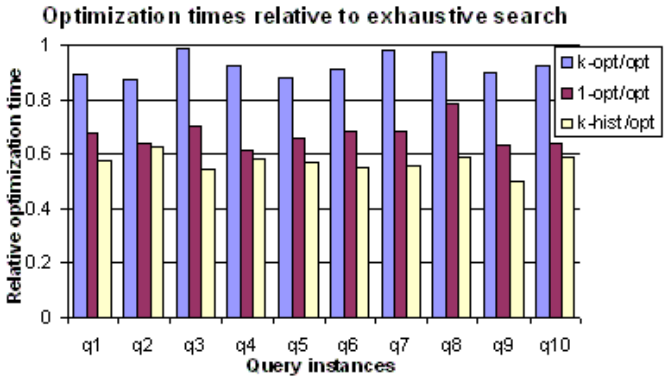

Figure 3. Average optimization time for different queries with four predicates

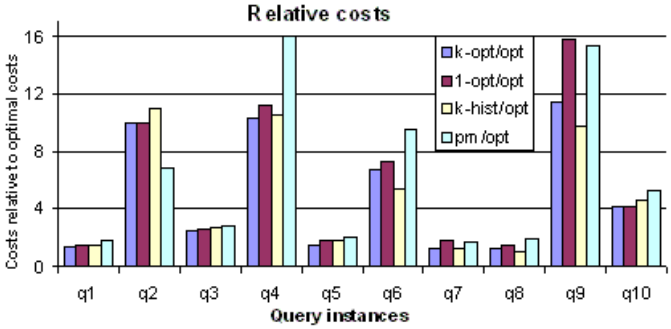

Figure 4. Relative cost optimality of various algorithms

Complexity. Figure 3 shows execution cost of various algorithms for different queries[3]. Since the optimal algorithm performs an exhaustive search, it requires the highest execution time. The predicate migration considers only one plan per level of join and hence the search space is very small. Therefore, it performs well in terms of execution time (not shown in the figure). 1-opt also performs optimization quickly (in the same order of the predicate migration) as it prunes the search space to the greatest level possible and thus time for enumerating the entire search space is reduced. K-opt searches a larger search space and hence performs slower than 1-opt and faster than optimal algorithms. Most interestingly, the k-hist algorithm, although it maintains more alternatives than the 1-opt, performs faster than 1-opt. The reason for this gain is the fact that k-hist reduces the search space by increasing the granularity of the space and approximating the results at every stage. Thus, a considerable saving in execution times is achieved.

[3] Since in this paper we do not concentrate on the question of whether different algorithms work better for different queries, we do not list the actual queries.

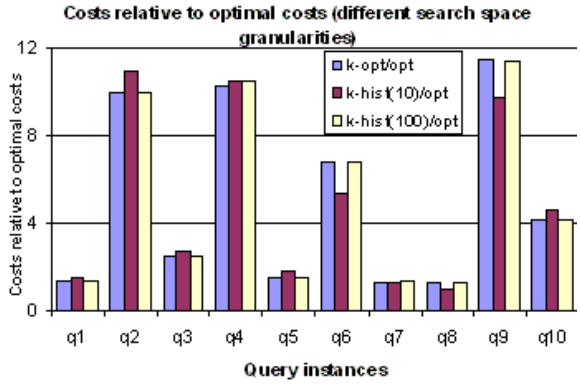

Figure 5. The effect of granularity on optimality

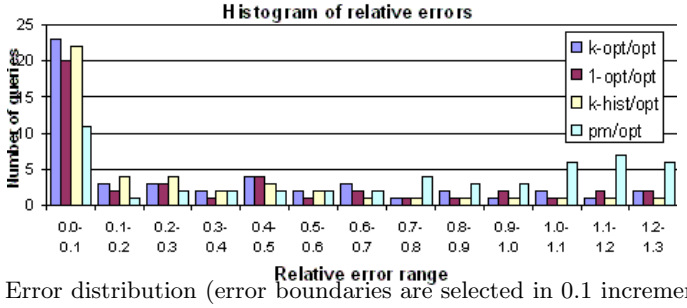

Figure 6. Error distribution (error boundaries are selected in 0.1 increments)

Degree of Optimality Figure 4 shows the relative behavior of different algorithms in terms of plan costs. The x-axis shows the queries and y-axis shows the ratio of the cost of each resulting plan to the cost of the cheapest plan (found by an optimal algorithm). According to this figure, predicate migration over-prunes the search space and the costs deviate from the optimal largely. The 1-opt, k-opt, and k-hist (granularity 10) algorithms perform relatively well. Most interestingly, k-hist performs almost as good as k-opt, and in some cases, it outperforms k-opt. Since, both are heuristic algorithms (each pruning the search space in different ways) this is not impossible. Since predicate migration does not capture variations in sizes, it behaves less predictably, choosing poor execution plans when different executions have varying sizes.

Figure 5 shows the effect of opt-histogram granularity on the optimality of the algorithms. As the granularity increases, k-hist resembles the k-opt algorithm. We also observed the degree of optimality using an histogram of the error, $\frac{Cost - Optimal\_cost}{Optimal\_cost}$ (Figure 6). In this figure, each bucket on the x-axis corresponds to an error range of size 0.1 (that is, intervals correspond to the number of queries which fall in each bucket.

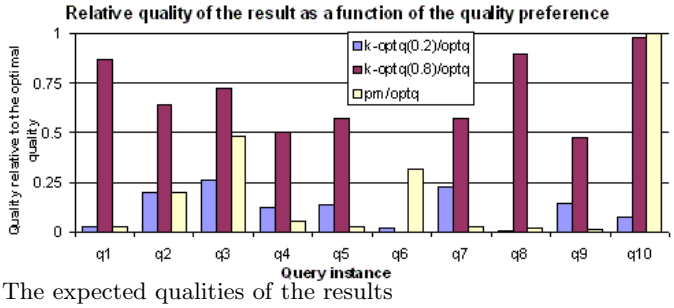

Figure 7. The expected qualities of the results

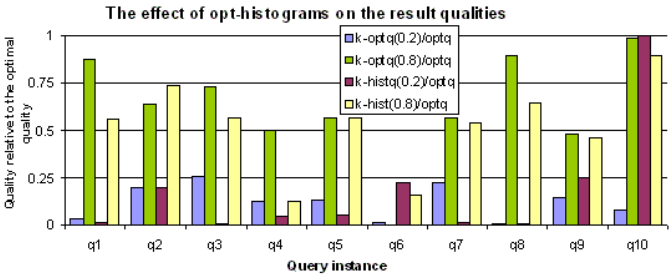

Figure 8. The effect of granularity on the result qualities

According to this figure, k-opt algorithm has the highest number of queries with the correct cost estimation (relative error range within the 0.0-0.1 bucket), whereas predicate migration has the fewest queries in this range. In fact, if we look at the larger error ranges, we see that predicate migration has relatively more queries in those buckets. Note also that, surprisingly, k-hist algorithm seems to cause a smaller number of large errors than the k-opt algorithm. Although, this is interesting, we do not believe that it is generalizable.

Quality-based Optimization Figure 7 shows that as the quality preference of the user increases, final qualities of the resulting query plans are closer to optimal. Since predicate migration does not consider qualities, as expected, its quality performance is unpredictable. Figure 8 shows that although the quality behavior of k-hist resembles that of k-opt in most cases, increased granularity causes less predictability.

6. Related Work

Selinger *et al.* [7] model the cost of a join assuming result sizes are independent of query execution order. Hellerstein [3] presents various techniques for efficiently optimizing queries that contain expensive methods. The first step is construction of a plan tree. Then the paths on the tree (streams) are sorted based on a rank[4]. Hellerstein discusses

[4] The definition of rank of a predicate is $Rank = \frac{(Selectivity-1)}{Cost}$

four strategies: pushing down predicates based on their ranks, pulling the costly sub-plans up, pulling expensive predicates based on join ranks rather than predicate ranks, and predicate migration, which considers the least costly plan at every stage of join. LDL [10], on the other hand, treats predicates as relations. One major drawback of this approach is over eager pull up decisions. Another drawback of this approach is that it is exponential in terms of number of relations and number of expensive predicates. [8, 11] compare LDL and predicate migration approaches. Their research focuses on pruning by ranks where $Rank = \frac{Cost}{(1-Selectivity)}$. [8] presents an algorithm that performs well with the assumption that predicates with smaller ranks are always applied earlier than others. The assumption however holds well for restrictive predicates only (predicates where all the parameters are bound), which ceases to be the case when varying binding patterns are used. Another assumption is that there is only one rank value per predicate. Authors propose pullrank and pushdown techniques in different contexts. [6, 12, 13] consider the problem of query optimization in the presence of limited binding patterns and provide efficient heuristics and optimization algorithms. However, none considers the cases where different binding patterns prune the data space differently (as it is the case when different binding patterns are implemented using different top-k or threshold criteria), leading to different number of tuples in the query result. Although [14] considers varying source coverages within an information integration framework, their cost and optimization models do not consider different binding patterns with different data coverages.

Recently, there are efforts [2, 5, 15, 16] addressing the fuzzy and probabilistic nature of database applications and benefit from this nature in top-k retrieval [1, 4]. Most of the existing work in this area concentrate on identification of efficient query processing techniques for queries with fuzzy predicates. [4] converts top-k retrieval queries to threshold (filtering) queries that can be processed using more traditional database processors. [17] looks at the effect of top-k evaluations on the query optimization task. In [15], we presented an approximate query evaluation algorithm that builds on [2, 16] to address the existence of non-progressive fuzzy predicates. The proposed approach minimizes the unnecessary accesses to non-progressive predicates, while providing error-bounds on the top-k retrieval results.

7. Conclusion

We presented a new query optimization model for multimedia and web databases. This model takes into account different bindings for every predicate, and it recognizes the fact that a query may have different

number of results and qualities depending on the query execution plan. The proposed approach further addresses the complexity of the optimization problem by *adaptively* reducing the granularity of the search space. For this purpose, unlike the data histograms which capture the data distribution, we use opt-histograms that capture the distribution of sub-query-plan values over many query optimization tasks.

References

1. S. Chaudhuri and L. Gravano, *Evaluating Top-k Selection Queries*, VLDB 1999, pp 397-410, 1999.
2. R. Fagin. *Fuzzy Queries in Multimedia Database Systems*, Principles of Database Systems, Seattle, WA, 1998.
3. J.M. Hellerstein, *Optimization Techniques for Queries with Expensive Methods,* Association of Computing Machinery, Transactions on Database Systems, Vol.23, No.2, June 1998, pp 113-157.
4. S. Chaudhuri and L. Gravano, *Optimizing Queries over Multimedia Repositories*, SIGMOD 1996, pp. 91-102, Canada, June 1996.
5. M. Ortega *et al.*, *Supporting Ranked Boolean Similarity Queries in MARS*, TKDE, (10)6, pp. 905-925, 1998.
6. D. Florescu, A. Levy, D. Suciu, and I. Manolescu. *Query Optimization in the Presence of Limited Access Patterns.* ACM SIGMOD, pp. 311-322, 1999.
7. P.G. Selinger *al.*, *Access Path Selection in a Relational Database Management System*, SIGMOD 1979, pp.23-34, 1979.
8. S. Chaudhuri and K. Shim, *Optimization of Queries with User-Defined Predicates,* VLDB 96, pp. 87-98, Bombay, India, 1996.
9. K.S. Candan and W.-S. Li. *On Similarity Measures for Multimedia Database Applications*, Knowledge and Information Systems 3(1): 30-51, 2001.
10. D. Chimenti, R. Gamboa, and R. Krishnamurthy. *Towards on Open Architecture for LDL,* VLDB 1989, pp. 195-203, 1989.
11. S. Chaudhuri, *An Overview of Query Optimization in Relational Systems*, Principles Of Database Systems, 1998.
12. R. Yerneni, C. Li, J. D. Ullman, and H. Garcia-Molina. *Optimizing Large Join Queries in Mediation Systems.* In ICDT, pp. 348-364, 1999.
13. V. Zadorozhny, L. Raschid, and M.E. Vidal. *Efficient Evaluation of Queries in a Mediator for WebSources,* to be published in SIGMOD 2002.
14. Z. Nie and S. Kambhampati. *Joint Optimization of Cost and Coverage of Query Plans in Data Integration.* In ACM CIKM, Atlanta, Georgia, November 2001.
15. K.S. Candan, W.-S. Li, and M.L. Priya. *Similarity-based Ranking and Query Processing in Multimedia Databases.* DKE 35(3): 259-298, 2000.
16. R. Fagin. *Combining Fuzzy Information from Multiple Systems.* 15^{th} ACM Symposium on Principles of Database Systems, pp. 216-226, June 1996.
17. D. Donjerkovic and R. Ramakrishnan, *Probabilistic Optimization of Top n Queries.* VLDB 1999, pp. 411-422. 1999.
18. L.P. Mahalingam and K.S. Candan. *Query Optimization in the Presence of Top-k Predicates*, Multimedia Inform. Systems Workshop, Capri, Italy, Nov. 2001.