Optimizing Queries over Multimedia Repositories

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Luis Gravano
Stanford University
gravano@cs.stanford.edu

Abstract

Multimedia repositories and applications that retrieve multimedia information are becoming increasingly popular. In this paper, we study the problem of selecting objects from multimedia repositories, and show how this problem relates to the processing and optimization of selection queries in other contexts, e.g., when some of the selection conditions are expensive user-defined predicates. We find that the problem has unique characteristics that lead to interesting new research questions and results. This article presents an overview of the results in [1]. An expanded version of that paper is in preparation [2].

1 Query Model

In this section we first describe the model that we use for querying multimedia repositories. Then, we briefly review related models for querying text and image repositories.

1.1 Our Query Model

In our model, a multimedia repository consists of a set of multimedia objects, each with a distinct object identity. Each multimedia object has a set of attributes, like the date the multimedia object was authored, a free-text description of the object, and the color histogram of an image contained in the object, for example. To query such a repository, users specify a Boolean condition that all objects in the query result should satisfy (the *filter condition*), together with some expression that is used to rank these objects (the *ranking expression*).

The *filter condition* of a query consists of a set of Boolean *atomic conditions* connected together by ANDs and ORs. An atomic condition compares an attribute value of a multimedia object (e.g., the color histogram associated with an image in the object) with a given constant attribute value (e.g., a given color histogram). However, such comparison differs from evaluation of a condition in a traditional selection expression in two ways. First, the comparison is a type-specific method. Unlike in a relational system, where the comparison between two values of the same built-in types is an inexpensive simple predicate, the comparison of two multimedia values can now be *expensive* (e.g., when comparing two

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Work done while the authors were at Hewlett-Packard Laboratories, Palo Alto, CA.

color histograms). Second, rarely does a user expect a multimedia object to match a given attribute value exactly. Rather, users are interested in the *grade of match* with which objects match the given attribute values [3]. Thus, given an object o, an attribute *attr*, and a constant *value*, the associated grade of match *Grade(attr, value)(o)* is a real number between 0 and 1, and expresses how well *o.attr* matches *value*. Then, an atomic filter condition is not necessarily an exact equality condition between two values (e.g., between a given color histogram h and the color histogram *oid.color_hist* of an object), but instead an inequality involving the grade of match between the two values together with some target grade. For example, *Grade(color_hist, h)(oid)* > 0.7 is an atomic filter condition that is satisfied by all objects whose color histogram matches the given color histogram h with a grade of match higher than 0.7.

Traditional selection queries ask for *all* tuples that match the selection condition, perhaps ordered using the values of a column, or a user-defined function [4]. However, the process of querying and browsing over a multimedia repository is likely to be interactive, and users will tend to ask for only "a few best matches" according to a ranking criterion. Therefore, a query in our model contains a *ranking expression* [3, 5] in addition to the filter condition that we described above. The ranking expression of a query assigns an order to each object in the repository that satisfies the filter condition in the query. In particular, the ranking expression can be a comparison that assigns a grade to each object. These grades are used to sort the objects, so that users can ask for the top 10 objects in the repository by a rank, for example. Ranking expressions can be atomic (e.g., *Grade(color_hist, h)(oid)*), or complex. Complex ranking expressions are built from atomic ones by using the *Min* and *Max* composition functions.

Putting everything together, we use the following SQL-like syntax to describe the queries in our model:

```
SELECT oid
FROM Repository
WHERE Filter_condition
ORDER [k] by Ranking_expression
```

Such a query asks for k objects in the object repository with the highest grade for the ranking expression, among those objects that satisfy the filter condition. The filter condition eliminates unacceptable matches, while the ranking expression orders the acceptable objects.

Example 1: *Consider a multimedia repository of information on criminals. A record on every person on file consists of a textual description* `p` *(for profile), a scanned fingerprint* `f`*, and a recording of a voice sample* `v`*. Given a target fingerprint* `F` *and voice sample* `V`*, the following example asks for records (1) whose fingerprint matches* `F` *with grade 0.9 or higher, or (2) whose profile matches the string* `'on parole'` *with grade 0.9 or higher and whose voice sample matches* `V` *with grade 0.5 or higher. The ranking expression ranks the acceptable records by the maximum of their grade of match for the voice sample* `V` *and for the fingerprint* `F`*. The answer contains the top 10 such acceptable records. (For simplicity, we omitted the parameter* `oid` *in the atomic conditions below.)*

```
SELECT oid
FROM   Repository
WHERE  (Grade(v, V) >= .5 AND Grade(p, 'on parole') >= .9)
        OR Grade(f, F) >= .9
ORDER [10] BY Max(Grade(f, F), Grade(v, V))
```

An interesting expressivity question is whether we actually need both the filter condition F and the ranking condition R. In other words, we would like to know whether we can "embed" the filter condition F in a new ranking expression R_F such that the top objects according to R_F are the top objects for R that satisfy F. In [1], we show that the above is not possible. In other words, our query model would be less expressive without filter conditions.

1.2 Related Query Models

Relational query models do not support ranking and grades of match in the sense of Section 1.1. On the other hand, extensible architectures of the *universal servers* can be exploited to support our proposed query model to some extent. An example of how to exploit an extensible architecture is Chabot [6], an image server based on Postgres. Chabot indirectly manages non-exact matches of, say, color histograms through a user-defined predicate *MeetsCriteria*. Given a color histogram h and some "criterion" *Mostly Red*, the predicate *MeetsCriteria("Mostly Red", h)* holds if histogram h is "sufficiently red," according to some hard-coded specification. Unfortunately, the Chabot approach is inflexible in that an explicit user-defined predicate would have to be created to handle ranking expressions that involve several attributes (e.g., to combine grades of match involving the image color histogram and the text caption of a multimedia object). However, an interesting open question is to study how and to what extent we can exploit the capabilities of an extensible architecture like that of Postgres to implement our proposed query model.

The models developed by the information retrieval community to query text repositories support ranking extensively [7]. In particular, the *vector-space* retrieval model typically uses lists of words as queries. Given a list of words, each document is assigned a grade of match for the query, which expresses how *similar* the document and the query are. Then, the documents are sorted based on this grade of match. In this model, both documents and queries are viewed as weight vectors, with one weight per word in the vocabulary. The weight associated with a word and a document (or query) is generally determined by the number of times that the word appears in the document and in the repository where the document occurs. The most common way to compute the grade of match of a document and a query is by taking the inner product of their corresponding weight vectors.

2 Storage Level Interface

So far we have defined our query model without describing the behavior and interface of the multimedia repositories. A repository has a set of multimedia objects. We assume that each object has an id and a set of attribute values, which we can only access through indexes. Given a value for an attribute, an index supports access to the ids of the objects that match that value with a certain grade, as we discuss below. Indexes also support access to the attribute values of an object given its oid.

Atomic filter conditions evaluate to either true or false, as discussed in the previous section. We assume that repositories support a *search* interface, denoted by *GradeSearch(attribute, value, min_grade)*. This call returns the ids of the objects that match the given value for the specified attribute with grade at least *min_grade*, together with the grades for the objects.

Also, repositories support a *probe* interface, denoted by *Probe(attribute, value, {oid})*. Given a set of object ids and a value for an attribute, this call returns the grade of each of the specified objects for the attribute value.

3 Query Processing

In this section we discuss how to process queries in the model defined above. In Section 3.1 we describe the processing of queries consisting of just a filter condition, which has close ties with traditional query processing. Later, we consider queries with both a filter condition and a ranking expression. How do we process such a query, given that we require only the top k matches? Can we use k to form a "pseudo-selection" condition that can be exploited for processing the entire query? This indeed is the case and we describe this novel approach to query processing in Section 3.2.

3.1 Processing Filter Expressions

The problem of optimizing queries that consist of just a filter condition (i.e., with no ranking expression) is indeed a "traditional" problem. Past work has addressed this problem only for some important special cases. However, a more general solution is needed in the context of multimedia selection queries.

For simplicity, we begin by considering the execution space and optimization of a *conjunctive* filter condition, i.e., a filter condition that is the conjunction of a set of atomic conditions. One way to determine the objects that satisfy such a filter condition is to scan the list of all object ids and evaluate each atomic condition for each object using the *probe* interface. The best such plan can be determined by viewing the problem as that of ordering a set of expensive selection conditions. The optimal order is the same as the ascending order of the *rank* $\frac{c}{1-s}$ of each condition, where c is the cost of evaluating the atomic condition for one object, and s is the selectivity of the condition [8]. These plans require executing a *Probe* for each condition and for each object that has a chance to be in the final result. Therefore, for a large repository, this step can be prohibitively expensive.

If one or more of the conditions support a *search* interface, we can do an "indexed" search. For example, we can use one condition for *GradeSearch*, and evaluate the rest of the conditions for the objects selected by using *Probe*. The optimization algorithm consists of estimating, for each choice of condition to use for *GradeSearch*, the cost of *GradeSearch* and the subsequent probing costs. The algorithm picks the condition that provides the least overall cost as the condition to be used for *GradeSearch*.

What happens when the filter condition is not a conjunction of atomic conditions, but instead a condition with atomic conditions connected via ORs as well as ANDs? We can view such a filter condition as an *AND-OR tree* where the tree structure reflects the nesting of propositional operators. As in the case of conjunctive queries, we can scan the list of object ids and then evaluate the atomic conditions to determine the answer to the query. Unfortunately, unlike the case of conjunctive queries, the problem of ordering the evaluation of atomic conditions for an arbitrary propositional condition is intractable [9], hence the need to rely on one of the well known heuristics [10] (cf. [11]).

How can we exploit the *search* interface of repositories for processing arbitrary filter conditions? Unlike the case of conjunctive queries, it is not sufficient to use *GradeSearch* on only one atomic condition to avoid "scanning" all the objects. Thus, the problem of identifying the *search-minimal* condition sets arises. We define a set of conditions S in a query to be search minimal if every object that qualifies to be in the answer set must belong to the result of searching on one of the conditions in S. In other words, searching on all conditions of S guarantees that we do not need to do a "scan" of all the objects. Furthermore, if any of the conditions in S is removed, then searching on the reduced set is no longer sufficient to guarantee the completeness of the answer. Thus, a *search-minimal execution* of a filter condition f searches the repository using a search-minimal condition set m for f, and executes the following steps. (In the following, we assume independence of the atomic conditions in the filter condition.)

- Determine an optimal search-minimal condition set m.

- For each condition $a \in m$:
  - Search on a to obtain a set of objects S_a.
  - Probe every object in S_a with the residual condition $R(a, f)$ to obtain a filtered set S'_a of objects that satisfy f.
- Return the union $\bigcup_{a \in m} S'_a$.

Thus, we have two new challenges here, even in the limited context of the search-minimal execution space: (a) Given a filter condition f and a search condition a, what is the residual predicate $R(a, f)$ that an object retrieved via search on a must satisfy to be in the answer set? (b) How do we determine the optimal search-minimal condition set? Our paper [1, 2] provides polynomial time algorithms to answer both questions under broad assumptions of cost models. We illustrate the step of determining residual predicates in the example below. The algorithm to determine an optimal search-minimal condition set for an independent filter condition is in [1]. Intuitively, the algorithm traverses the AND-OR condition tree in a bottom-up fashion. For each AND node, it takes the optimal search-minimal condition set of one of the subtrees (the one with minimal cost), and for each OR node, it takes the union of the search-minimal condition sets of all its subtrees.

Example 2: *Consider the filter condition $f = a_4 \wedge ((a_1 \wedge a_2) \vee a_3)$. The residue of the atomic condition a_2 is $R(a_2, f) = a_1 \wedge a_4$. As another example, $R(a_4, f) = (a_1 \wedge a_2) \vee a_3$. Then, any object that satisfies a_4 and also satisfies $R(a_4, f)$ satisfies the entire condition f. Observe that each of $\{a_4\}$, $\{a_2, a_3\}$, and $\{a_1, a_3\}$ is a search-minimal condition set. If we decide to search on $\{a_2, a_3\}$, the following three steps yield exactly all of the objects that satisfy f:*

1. *Search on a_2 and probe the retrieved objects with residue $R(a_2, f) = a_1 \wedge a_4$. Keep the objects that satisfy $R(a_2, F)$.*
2. *Search on a_3 and probe the retrieved objects with residue $R(a_3, f) = a_4$. Keep the objects that satisfy $R(a_3, F)$.*
3. *Return the objects kept.*

The discussion above can be cast in a more traditional framework. We investigated the problem of finding an optimal execution for a selection in the presence of indexes as well as user-defined methods for arbitrary propositional clauses. We focussed on search-minimal executions, which correspond to using a minimal number of indexes to avoid sequential scans of the data. Thus, when filter conditions are a conjunction of literals, a search-minimal execution corresponds to using a single index. We showed that we needed new algorithms to handle arbitrary filter conditions even in the context of search-minimal executions. Further complexity arises if we step beyond the search-minimal execution space. This corresponds to doing index intersections to evaluate a query (e.g., for processing a conjunctive filter condition). In this case, our problem would become closely related to that of query processing with index AND-ing and OR-ing [12]. However, our model would require that we extend the existing results, since *Probe* costs are not zero anylonger. Furthermore, not only would we need to choose the superset of a search-minimal set to search, but we should also address the more complex ordering of search-result merges and probes [2].

3.2 Processing Ranking Expressions

We now look at queries consisting only of ranking expressions. Such queries have the following form:

```
SELECT oid
FROM Repository
ORDER [k] by Ranking_expression
```

The result of this query is a list of k objects in the repository with the highest grade for the given ranking expression. The ranking expressions are built from atomic expressions that are combined using the *Min* and *Max* operators (Section 1.1).

At first glance, processing such queries appears to be troublesome. Although we need no more than k best objects, we must retrieve each object, evaluate the ranking expression over each object, and then sort the objects accordingly. In other words, although we are interested in only the top k objects, we are unable to take advantage of k for query processing. We show that we can map a given ranking expression into a filter condition, and process the ranking expression "almost" as if it were a filter condition. This result makes it possible to take advantage of the parameter k for query processing and is central to processing queries with ranking expressions using the techniques of Section 3.1 for filter conditions.

Fagin presented a novel approach to take advantage of k in processing a query consisting of a ranking expression such as $R = Min(a_1, \ldots, a_n)$, where the a_i's are independent atomic expressions [3]. Fagin has proved the important result that his algorithm to retrieve k top objects for an expression R that is a *Min* of independent atomic expressions is asymptotically optimal with arbitrarily high probability in terms of the number of objects retrieved.

Although Fagin's strategy helps reduce the number of objects that need to be retrieved to process a ranking expression, it cannot treat a ranking expression as a selection condition from the point of view of query processing (e.g., to determine the search-minimal condition set). Our key contribution is not only to take advantage of k (which Fagin did) but also to view a ranking expression as a filter expression so as to make query processing and cost-based optimization of queries uniform irrespective of the presence of ranking expressions.

When the query contains both a filter condition F and a ranking expression R, it asks for k top objects by the ranking expression R that satisfy F. Using the results in this section, we can translate this query into the problem of optimizing the filter condition $F \wedge F'$, where F' is the filter condition "associated" with R. We now describe how such an F' is determined.

Given a ranking expression R and the number k of objects desired, we show that:

1. There is an algorithm to assign a grade to each atomic expression in R, and a filter condition F with the same "structure" as R, such that F is expected to retrieve at least the top k objects according to R.

2. There is a search-minimal execution for F that retrieves an expected number of objects that is no larger than the expected number of objects that Fagin's algorithm would retrieve for R and k.

Example 3: *Consider a ranking expression $e = Min(Grade(a_1, v_1), Grade(a_2, v_2))$, where a_i is an attribute, and v_i a constant value. We want two objects with the top grades for e. Now, suppose that we can somehow find a grade G (the higher the better) such that there are at least two objects with grade G or higher for expression e. Therefore, if we retrieve all of the objects with grade G or higher for e, we can simply order them according to their grades, and return the top two as the result to the query.*

In other words, we can process e by processing the following associated filter condition *f, followed by a sorting step of the answer set for f:*

$$f = (Grade(a_1, v_1)(o) \geq G) \wedge (Grade(a_2, v_2)(o) \geq G)$$

By processing f using the strategies in Section 3.1, we obtain all of the objects with grade G or higher for a_1 and v_1, and for a_2 and v_2. Therefore, we obtain all of the objects with grade G or higher for the ranking expression e. If there are enough objects in this set (i.e., if there are at least two objects), then we know we have retrieved the top objects that we need to answer the query with ranking expression e. Similarly, we can process a ranking expression $e' = Max(Grade(a_1, v_1), Grade(a_2, v_2))$ as filter condition $f' = (Grade(a_1, v_1)(o) \geq G') \vee (Grade(a_2, v_2)(o) \geq G')$, for some grade G'.

The example above shows how we can process a ranking expression e as a filter condition f followed by a sorting step. But the key point in mapping the ranking problem to a (modified) filtering problem is finding the grade G to use in f. In [1], we present the algorithm `Grade_Rank`, which given the number of objects desired k, a ranking expression e, and selectivity statistics, produces the grade G for the filter condition f.

Our approach allows us to translate the ranking expressions into filter conditions, and to use the processing strategy of Section 3.1. At least k objects are expected to satisfy F. However, if at run time we find that fewer than k objects satisfy F, we should lower the grade G used in F. We will investigate strategies to lower G as part of our future work.

It is natural to ask how our algorithm compares with Fagin's in terms of retrieval efficiency. In [1], we show that if we process a ranking expression (and its associated number k of objects requested) by using a filter condition F with grade G as determined by algorithm `Grade_Rank`, we can expect to retrieve no more objects than Fagin's algorithm, under some assumptions on the repositories. Furthermore, our experiments show that the approach that we outlined in this section still is a desirable one when the assumptions on the repository do not hold strictly.

Although in this section we showed how to process a ranking expression like a filter condition, the semantics of both the filter condition and the ranking expression remain distinct. (See Section 1.1.) After processing a ranking expression as a filter condition, we have to compute the grade of the retrieved objects for the ranking expression, and sort them before returning them as the answer to the query.

3.3 Putting our Results in Perspective

Consider queries that do not have a ranking expression. Such queries consist of a filter condition with AND and OR Boolean connectives where some of the atomic conditions can be expensive. Our results on processing such queries generalize past work on processing conditions with expensive predicates. In effect, our algorithms consider the case where both of the following conditions are true: (a) the filter condition is more general than a conjunction of literals, and (b) an expensive predicate can be evaluated by using *GradeSearch* (indexed access) as well as by using *Probe* (both the arguments of the predicate are bound). Although past work on expensive predicates addresses the case where only condition (b) holds [13, 14], it does not address the case where condition (a) holds as well. Finally, note that the algorithms and results of the previous section are completely *independent* of the nature of the atomic predicates as long as a selectivity and a cost measure are available.

Several interesting query processing questions remain open. The search-minimal execution space is restrictive in a way analogous to using only one index for processing a traditional selection condition. Eliminating the restriction that executions be search minimal opens interesting directions to explore.

Our ranking expressions are built using the *Min* and *Max* operators. Another interesting question to explore is how to process ranking expressions that use operators like a weighted sum, for example, and deciding what other operators would be useful to allow in ranking expressions.

References

[1] Surajit Chaudhuri and Luis Gravano. Optimizing queries over multimedia repositories. In *Proceedings of the 1996 ACM SIGMOD Conference on Management of Data*, pages 91–102, Montreal, June 1996.

[2] Surajit Chaudhuri and Luis Gravano. Optimizing queries over multimedia repositories. Technical report, 1996. In preparation.

[3] Ronald Fagin. Combining fuzzy information from multiple systems. In 15^{th} *ACM Symposium on Principles of Database Systems*, June 1996. Also available as IBM Almaden Research Center Technical Report RJ 9980.

[4] Jim Gray et al. Data cube: A relation aggregation operator generalizing group by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1), 1996.

[5] F. Rabitti. Retrieval of multimedia documents by imprecise query specification. In *Proceedings of the 1990 EDBT*, Venice, March 1990.

[6] Virginia E. Ogle and Michael Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28(9), September 1995.

[7] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison Wesley, 1989.

[8] J. M. Hellerstein and M. Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the 1993 ACM SIGMOD Conference on Management of Data*, Washington D. C., May 1993.

[9] L. T. Reinwald and R. M. Soland. Conversion of limited entry decision tables to optimal computer programs I: Minimum average processing time. *Journal of the ACM*, 13(3), 1966.

[10] A. Kemper, G. Moerkotte, and M. Steinbrunn. Optimizing boolean expressions in object bases. In *Proceedings of the 18th VLDB Conference*, Vancouver, August 1992.

[11] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn. Optimizing disjunctive queries with expensive predicates. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*, Minneapolis, May 1994.

[12] C. Mohan. Single table access using multiple indexes: Optimization, execution and concurrency control techniques. In *Proceedings of the 1990 EDBT*, Venice, March 1990. Also available as IBM Almaden Research Center Technical Report RJ 7341.

[13] Surajit Chaudhuri and Kyuseok Shim. Optimization of queries with user-defined predicates. In *Proceedings of the 22nd VLDB Conference*, Bombay, India, September 1996.

[14] J. Hellerstein. Optimization techniques for queries with expensive methods. Technical Report CS-TR-96-1304, University of Wisconsin, Madison, February 1996. Ph.D. dissertation.