Fuzzy Queries in Multimedia Database Systems

Ronald Fagin
IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099

email: fagin@almaden.ibm.com
http://www.almaden.ibm.com/cs/people/fagin/

Invited paper: Proc. 1998 ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems

Abstract

There are essential differences between multimedia databases (which may contain complicated objects, such as images), and traditional databases. These differences lead to interesting new issues, and in particular cause us to consider new types of queries. For example, in a multimedia database it is reasonable and natural to ask for images that are somehow "similar to" some fixed image. Furthermore, there are different ways of obtaining and accessing information in a multimedia database than information in a traditional database. For example, in a multimedia database, it might be reasonable to have a query that asks for, say, the top 10 images that are similar to a fixed image. This is in contrast to a relational database, where the answer to a query is simply a set. (Of course, in a relational database, the result to a query may be sorted in some way for convenience in presentation, such as sorting department members by salary, but logically speaking, the result is still simply a set, with a crisply-defined collection of members.) In this paper, we survey some new issues that arise for multimedia queries, with a particular focus on recent research by the author, developed in the context of the Garlic system at the IBM Almaden Research Center.

1 Introduction

As hardware becomes more powerful and as software becomes more sophisticated, it is increasingly possible to make use of multimedia data, such as images and video. If we

wish to access multimedia data through a database system, a number of new issues arise. In particular, objects in a multimedia database may be much more complicated than a typical entry in a column in a relational database. For example, a multimedia database might deal with pictures that have a complicated coloring pattern and that contain a number of shapes. These differences lead to fundamental differences between multimedia queries and standard relational database queries. In this paper, we consider some of these issues, along with existing solutions and open problems.

In Section 2, we discuss the notion of an atomic query in a multimedia database. Here we are often interested in "approximate matches". Therefore, an atomic query in a multimedia database is typically much harder to evaluate than an atomic query in a relational database. In Subsection 2.1, we discuss methods of speeding up the evaluation of atomic multimedia queries. In Section 3, we consider the evaluation of Boolean combinations of atomic queries. Unlike the situation in relational databases, where the semantics of a Boolean combination is quite clear, in multimedia databases it is not at all clear what the semantics is of even the conjunction of atomic queries. In order to make sense of this notion, it is convenient to introduce "graded" (or "fuzzy") sets, in which scores are assigned to objects, depending on how well they satisfy atomic queries. There are then "aggregation functions", which combine scores (under subqueries) for an object into an overall score (under the full query) for that object. In Section 4, we consider the fact that a system capable of dealing with various flavors of multimedia data is often actually middleware, and we discuss the impact of this on multimedia queries. For example, it may be possible to obtain data from some multimedia repositories in only limited ways. Algorithms for query evaluation must, of course, respect these limitations. In Subsection 4.1, we discuss such an algorithm from [Fa96] that is optimal under certain conditions. This algorithm has been implemented in the Garlic system [CHS+95, CHN+95] at the IBM Almaden Research Center. In the process of the implementation, interesting "real-world" issues arose. These are discussed in Subsection 4.2. Finally, in Section 5, we discuss an issue that arises for multimedia queries but not for standard relational queries. Namely, in a multimedia query the user might want to give extra weight to certain subqueries. For example, the user might be interested in objects that are both red and round, but care more about the color than the shape. In Section 5, we discuss a technique, given by a formula from [FW97], that shows how to do this weighting. This formula is the unique one that satisfies certain natural properties. Finally, in Section 6, we summarize, and give open problems.

2 Atomic Queries

In a multimedia system, an atomic query might ask for an object that is red. For simplicity, in this paper we will write such an atomic query as *Color='red'*, although in an actual system this query might be expressed by selecting a color from a color wheel, or by selecting an image I (that might be predominantly red) and asking for other images whose colors are "close to" that of image I. In response to a query, a multimedia system

might typically return a sorted list of the top, say, 10, items in its database that match the query the best. For example, if the query asks for red objects, then the result would be a sorted list with the reddest object first, the next reddest object second, etc.

In order to decide which object is reddest, which object is second reddest, and so on, there must be a notion of closeness of color, so that the reddest object is the object whose color is closest to red. Multimedia systems have sophisticated color-matching algorithms [Io89, NBE+93, SO95, SC96] that compute the closeness of the colors of two images. For example, an image that contains a lot of red and a little green might be considered moderately close in color to another image with a lot of pink and no green.

Computing the closeness in color between two images may be computationally expensive. For example, we now describe a method for determining closeness in color that was suggested by Ioka [Io89], and that is implemented in the QBIC[1] [NBE+93] system developed at the IBM Almaden Research Center. As described in [FBF+94], each object has a k-element color histogram (typical values of k are 64, 100, or 256). Let $\mathbf{x}$ and $\mathbf{y}$ be two k-dimensional vectors that represent the color histograms of two objects. The color distance between the two objects is taken to be the distance between the histograms, which is defined to be

$$\sqrt{(\mathbf{x}-\mathbf{y})^T A(\mathbf{x}-\mathbf{y})}. \tag{1}$$

Here the superscript T indicates taking the transpose, and A is a (symmetric) matrix whose (i,j)th entry describes the similarity between color i and color j.

As with colors, there are a number of ways to define closeness between shapes. These include methods based on turning angles [ACH+90, MKC+91], on the Hausdorff distance [HRK92], on various forms of moments [KK97, TC91], and on Fourier descriptors [Ja89]. Mehtre, Kankanhalli and Lee [MKL97] and Mumford [Mu91] discuss and compare various approaches.

2.1 Speeding Up the Evaluation

As we have discussed, an attribute (such as color or shape) of a multimedia object is often represented by a multidimensional vector. This suggests the use of a multidimensional indexing method, in order to speed up the evaluation of atomic multimedia queries. But multimedia data often have high dimensionalities: for example, as we noted, there are commonly 64 or more color features. This is sometimes called the "dimensionality curse" [FBF+94, page 244]. Two popular multidimensional indexing methods, namely linear quadtrees [Sa89] and grid files [NHS84], grow exponentially with the dimensionality. So these methods are not practical in these situations. Another popular multidimensional indexing method is R*-trees [BKSS90]. These tend to be more robust for higher dimensions, at least for dimensions up to around 20 [Ot92]. It is the author's opinion that much more work is needed in high-dimensional indexing, or similar techniques, in order to deal effectively with the hard issues of efficiently evaluating atomic multimedia queries.

[1] QBIC, which stands for Query By Image Content, is a trademark of IBM Corporation.

We now discuss one approach to help deal with the dimensionality curse. As a concrete example, we focus on a method from [HSE+95] that is used to deal with color feature vectors, and that is referred to in [FBF+94] as a *distance-bounding strategy*. Let $d(\mathbf{x}, \mathbf{y})$ be the distance between color histograms $\mathbf{x}$ and $\mathbf{y}$ (for example, $d(\mathbf{x}, \mathbf{y})$ could be given by (1)). They associate with each (long) color feature vector $\mathbf{x}$ a short (say, dimension 3) color vector $\hat{\mathbf{x}}$ that, intuitively, "summarizes" $\mathbf{x}$. They then give a simple-to-compute distance measure $\hat{d}$ that gives the distance between small color vectors, and show that if $\mathbf{x}$ and $\mathbf{y}$ are color feature vectors, then

$$d(\mathbf{x}, \mathbf{y}) \geq \hat{d}(\hat{\mathbf{x}}, \hat{\mathbf{y}}). \tag{2}$$

Thus, assume that we wish to find those objects whose color feature vector $\mathbf{y}$ is close to some target color feature vector $\mathbf{x}$ (say, using (1)). We see from (2) that we can restrict our attention to objects whose short color vector $\hat{\mathbf{y}}$ is close to the short color vector $\hat{\mathbf{x}}$. Intuitively, $\hat{d}$ is being used as a "filter" to eliminate from consideration objects with a short color vector $\hat{\mathbf{y}}$ where $\hat{d}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is too large. This could be useful in two ways. First, we could potentially have a multidimensional index on short color vectors. Second, even without an index, we often need only compute $\hat{d}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ rather than $d(\mathbf{x}, \mathbf{y})$; the former is much easier to compute.

We now describe another approach, that is useful for situations where the user is browsing, that is, searching for objects in the database that are close (say, in color) to that of another image in the database. This approach takes advantage of the fact that in many multimedia database situations (such as with image databases), updates are done rarely, if at all [FBF+94, Gr97]. The idea is to precompute, for each image in the database, the top, say, 200 images in the database that are closest to it, and store the information with the image. If the user asks for those images whose color is close to the color of some other image in the database, no painful computation such as that given by the formula (1) needs to be done in real time.

3 Boolean Combinations of Atomic Queries

As we noted earlier, the result of a multimedia query is typically a sorted list. For example, if the query asks for red objects, then the result would be a sorted list with the reddest object first, the next reddest object second, etc. By contrast, the result of a query to a relational database is simply a set. This leads to a mismatch: the result of some queries is a sorted list, and for other queries, it is a set. How do we combine such queries in Boolean combinations? We now discuss the solution proposed in [Fa96], which has been implemented in Garlic, a multimedia information system being developed at the IBM Almaden Research Center.

As an example (given in [Fa96]), let us consider an application of a store that sells compact disks. A typical traditional database query might ask for the names of all albums where the artist is the Beatles. The result is a set of names of albums. A

multimedia query might ask for all album covers with a particular shade of red. Here the result is a sorted list of album covers. We see the mismatch in this example: the query *Artist='Beatles'* gives us a set, whereas the query *AlbumColor='red'* gives us a sorted list. How do we combine a traditional database query and a multimedia query? For example, consider the query

(Artist='Beatles') $\wedge$ *(AlbumColor='red').*

What is the result of this query? In this case, we probably want a sorted list, that contains only albums by the Beatles, where the list is sorted by goodness of match in color. What about more complicated queries? For example, what should the result be if we replaced $\wedge$ by $\vee$ in the previous query? Is the answer a set, a sorted list, or some combination? How about if we combine two multimedia queries? An example is given by the query

(Color='red') $\wedge$ *(Shape='round').*

The solution in [Fa96] is in terms of "graded" (or "fuzzy") sets [Za65]. A graded set is a set of pairs (x, g), where x is an object (such as a tuple), and g (the grade) is a real number in the interval $[0, 1]$. It is sometimes convenient to think of a graded set as corresponding to a sorted list, where the objects are sorted by their grades. Thus, a graded set is a generalization of both a set and a sorted list.

As earlier, we take *atomic queries* to be of the form $X = t$, where X is the name of an attribute, and t is a target. Examples are the atomic query *Artist='Beatles'* and the atomic query *AlbumColor='red'*. *Queries* are Boolean combinations of atomic queries.

For each atomic query, a grade is assigned to each object. The grade represents the extent to which that object fulfills that atomic query, where the larger the grade is, the better the match. In particular, a grade of 1 represents a perfect match. For traditional database queries, such as *Artist='Beatles'*, the grade for each object is either 0 or 1, where 0 means that the query is false about the object, and 1 means that the query is true about the object. For other queries, such as a multimedia query corresponding to *AlbumColor='red'*, grades may be intermediate values between 0 and 1.

We must now consider how to take Boolean combinations of atomic queries. A number of different rules for evaluating Boolean combinations in fuzzy logic have appeared in the literature. In particular, there are a number of reasonable "aggregation functions" that assign a grade to a fuzzy conjunction, as a function of the grades assigned to the conjuncts. An *m-ary aggregation function* is a function from $[0, 1]^m$ to $[0, 1]$. For the sake of generality, we will often consider m-ary aggregation functions for evaluating conjunctions of m atomic queries, although in practice an m-ary conjunction is almost always evaluated by using an associative 2-ary function that is iterated.

We consider now the standard rules of fuzzy logic, as defined by Zadeh [Za65]. If x is an object and Q is a query, let us denote by $\mu_Q(x)$ the grade of x under the query Q.

If we assume that $\mu_Q(x)$ is defined for each atomic query Q and each object x, then it is possible to extend to queries that are Boolean combination of atomic queries via the following rules.

Conjunction rule: $\mu_{A \wedge B}(x) = \min\{\mu_A(x), \mu_B(x)\}$

Disjunction rule: $\mu_{A \vee B}(x) = \max\{\mu_A(x), \mu_B(x)\}$

Negation rule: $\mu_{\neg A}(x) = 1 - \mu_A(x)$

Thus, the standard conjunction rule of two formulas in fuzzy logic is based on using min as the 2-ary aggregation function.

These rules are attractive for two reasons. First, they are a conservative extension of the standard propositional semantics. That is, if we restrict our attention to situations where $\mu_Q(x)$ is either 0 or 1 for each atomic query Q, then these rules reduce to the standard rules of propositional logic. The second reason is because of an important theorem of Bellman and Giertz [BG73], and extended and simplified by Yager [Ya82], Voxman and Goetschel [VG83], Dubois and Prade [DP84], and Wimmers [Wi98]. We now discuss the Bellman-Giertz theorem.

The standard conjunction and disjunction rules of fuzzy logic have the nice property that if Q_1 and Q_2 are logically equivalent queries involving only conjunction and disjunction (not negation), then $\mu_{Q_1}(x) = \mu_{Q_2}(x)$ for every object x. For example, $\mu_{A \wedge A}(x) = \mu_A(x)$. As another example, $\mu_{A \wedge (B \vee C)}(x) = \mu_{(A \wedge B) \vee (A \wedge C)}(x)$. This is desirable in a database context, since then an optimizer can replace a query by a logically equivalent query, and be guaranteed of getting the same answer.

Furthermore, the aggregation function min for conjunction is *monotone*, in the sense that if $\mu_A(x) \leq \mu_A(x')$, and $\mu_B(x) \leq \mu_B(x')$, then $\mu_{A \wedge B}(x) \leq \mu_{A \wedge B}(x')$. Similarly, the aggregation function max for disjunction is monotone. Monotonicity is certainly a reasonable property to demand, and models the user's intuition. Intuitively, if the grade of object x' under the query A is at least as big as that of object x, and the grade of object x' under the query B is at least as big as that of object x, then the grade of object x' under the query $A \wedge B$ is at least as big as that of object x.

The next theorem, due to Yager [Ya82] and Dubois and Prade [DP84], is a variation of the Bellman-Giertz theorem that says that min and max are the unique aggregation functions for conjunction and disjunction with these properties. (Bellman and Giertz's original theorem required more assumptions.)

Theorem 3.1: *The unique aggregation functions for evaluating $\wedge$ and $\vee$ that preserve logical equivalence of queries involving only conjunction and disjunction and that are monotone in their arguments are min and max.*

Many other aggregation functions have appeared in the literature. We now consider an important class of 2-ary aggregation functions. A *triangular norm* [SS63, DP80] is a 2-ary aggregation function t that satisfies the following properties:

$\wedge$-Conservation: $t(0,0) = 0$; $t(x,1) = t(1,x) = x$.

Monotonicity: $t(x_1, x_2) \leq t(x_1', x_2')$ if $x_1 \leq x_1'$ and $x_2 \leq x_2'$.

Commutativity: $t(x_1, x_2) = t(x_2, x_1)$.

Associativity: $t(t(x_1, x_2), x_3) = t(x_1, t(x_2, x_3))$.

It is reasonable to expect an aggregation function for conjunction to satisfy each of the properties of a triangular norm. We call the first condition "$\wedge$-conservation", since it implies in particular a conservative extension of the standard propositional semantics for conjunction, as we discussed in the case of min.

A *triangular co-norm* [DP85] is a 2-ary aggregation function s that satisfies monotonicity, commutativity, and associativity as above, along with the following boundary conditions:

$\vee$-Conservation: $s(1,1) = 1$; $s(x,0) = s(0,x) = x$.

It is reasonable to expect an aggregation function for disjunction to satisfy each of the properties of a triangular co-norm.

Triangular norms and triangular co-norms are duals, in the sense that if t is a triangular norm, then the function s defined by $s(x_1, x_2) = 1 - t(1 - x_1, 1 - x_2)$ is a triangular co-norm [Al85]. Bonnissone and Decker [BD86] show that for suitable negation aggregation functions n (such as the standard $n(x) = 1 - x$), the natural generalization of DeMorgan's Laws hold between a triangular norm t and its co-norm s:

$$s(x_1, x_2) = n(t(n(x_1), n(x_2)))$$

$$t(x_1, x_2) = n(s(n(x_1), n(x_2)))$$

The min function is a triangular norm, with co-norm the max function. Other triangular norms and their co-norms are discussed in [BD86, Mi89]; see also [Fa96] and Zimmermann's textbook [Zi96].

We can define an m-ary query (such as the conjunction of m formulas) in terms of an m-ary aggregation function. In [Fa96], the semantics of an m-ary query $F(A_1, \ldots, A_m)$ is given by defining $\mu_{F(A_1,\ldots,A_m)}$. For example, the standard fuzzy logic semantics of the conjunction $A_1 \wedge \ldots \wedge A_m$ is given by defining

$$\mu_{A_1 \wedge \ldots \wedge A_m}(x) = \min\{\mu_{A_1}(x), \ldots, \mu_{A_m}(x)\},$$

for each object x. Let t be an m-ary aggregation function. We define the m-ary query $F_t(A_1, \ldots, A_m)$ by taking

$$\mu_{F_t(A_1,\ldots,A_m)}(x) = t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x)).$$

For example, if t is min, then $F_t(A_1, \ldots, A_m)$ is equivalent in the standard fuzzy semantics to $A_1 \wedge \ldots \wedge A_m$, and if t is max, then $F_t(A_1, \ldots, A_m)$ is equivalent in the standard fuzzy semantics to $A_1 \vee \ldots \vee A_m$.

As we mentioned earlier, an m-ary conjunction can of course be obtained from a 2-ary conjunction by iterating. For example, if 2-ary conjunction is defined by the 2-ary aggregation function t, then 3-ary conjunction can be defined by $t(t(x_1, x_2), x_3)$. As discussed in [Fa96], the following two properties hold for every m-ary aggregation function that is obtained by iterating a triangular norm:

Strictness: $t(x_1, \ldots, x_m) = 1$ iff $x_i = 1$ for every i.

Monotonicity: $t(x_1, \ldots, x_m) \leq t(x'_1, \ldots, x'_m)$ if $x_i \leq x'_i$ for every i.

Thus, an aggregation function is strict if it takes on the maximal value of 1 precisely if each argument takes on this maximal value.

We call $F_t(A_1, \ldots, A_m)$ a *strict* (resp., *monotone*) *query* if t is strict (resp., monotone). The only properties of a query that are required for the theorems in [Fa96] (some of which we shall discuss shortly) are strictness and monotonicity. Strictness is needed for a lower bound on the efficiency of algorithms for evaluating queries under certain assumptions, and monotonicity for an upper bound.

There are aggregation functions for conjunction that have been considered in the literature that are not triangular norms. For example, Thole, Zimmermann, and Zysno [TZZ79] found various weighted and unweighted arithmetic and geometric means to perform empirically quite well. Such aggregation functions are not triangular norms: in fact, the arithmetic mean does not conserve the standard propositional semantics, since with arguments 0 and 1 it takes the value 1/2, rather than 0. These functions do satisfy strictness and monotonicity, and so the lower and upper bounds of [Fa96] hold even in this case. Thus, if a system were to use, say, the arithmetic mean as an aggregation function for evaluating the conjunction, then these lower and upper bounds tell us how efficiently we can expect to evaluate the conjunction under natural assumptions.

4 Middleware

Because of the many varieties of data that a multimedia database system must handle, such a system may often really be "middleware". That is, the system is "on top of" various subsystems, and integrates results from the subsystems. For example, the Garlic system of the IBM Almaden Research Center is such a middleware system. Garlic is designed to be capable of integrating data that resides in different database systems as well as a variety of non-database data servers. A single Garlic query can access data in a number of different subsystems. An example of a nontraditional subsystem that Garlic accesses is QBIC [NBE+93], which was mentioned earlier. QBIC can search for images by various visual characteristics such as color and texture.

There are a host of problems associated with middleware, including schema integration, dirty data (caused by multiple sources having conflicting information), security concerns, etc. We will focus here on some specific issues related to multimedia queries.

What can we assume about the interface between a multimedia database system (such as Garlic) and a subsystem such as QBIC? In response to a query, such as *Color='red'*, we can assume that the subsystem will output the graded set consisting of all objects, one by one, along with their grades under the subquery, in sorted order based on grade, until Garlic tells the subsystem to stop. Then Garlic could later tell the subsystem to resume outputting the graded set where it left off. Alternatively, Garlic could ask the subsystem for, say, the top 10 objects in sorted order, along with their grades, then request the next 10, etc. We refer to such types of access as "sorted access".

There is another way that we could expect Garlic to interact with the subsystem. Garlic could ask the subsystem the grade (with respect to a query) of any given object. We refer to this as "random access".

Because of the limited modes of access to subsystems, issues of efficient query evaluation in a middleware system are very different from those in a traditional database system. In fact, it is not even clear what "efficient" means in a middleware system.

We now give the performance cost of an algorithm, as defined in [Fa96]. This measure of cost corresponds intuitively to the amount of information that an algorithm obtains from the database. The *sorted access cost* is the total number of objects obtained from the database under sorted access. For example, if there are only two lists (corresponding, in the case of conjunction, to a query with two conjuncts), and some algorithm requests altogether the top 100 objects from the first list and the top 20 objects from the second list, then the sorted access cost for this algorithm is 120. Similarly, the *random access cost* is the total number of objects obtained from the database under random access. The *middleware cost* is taken to be $c_1 \cdot S + c_2 \cdot R$, where S is the sorted access cost, R is the random access cost, and where c_1 and c_2 are positive constants.[2] The middleware cost is not a measure of total system cost, since it ignores the costs inside of a "black box" like QBIC. There are situations (such as in the case of a query optimizer) where we want a more comprehensive cost measure. Finding such cost measures is an interesting open problem.

4.1 Algorithms for Query Evaluation

In this subsection, we give an algorithm from [Fa96] for evaluating monotone queries. Under certain assumptions the algorithm is optimally efficient up to a constant factor.

Probably the most important queries are those that are conjunctions of atomic queries. For the sake of the current discussion, let us assume for now that conjunctions are being

[2] In [Fa96], the middleware cost is taken for convenience to be simply $S + R$, the sum of the sorted access cost and the random access cost. It is then noted that these two notions of middleware cost (namely, $S + R$ and $c_1 \cdot S + c_2 \cdot R$) are within constant multiples of each other, and so the same results hold in the "big O" notation.

evaluated by the standard min rule. An example of a conjunction of atomic queries is the query

(Artist='Beatles') $\wedge$ *(AlbumColor='red').*

that we have discussed in our running example. In this example, the first conjunct *Artist='Beatles'* is a traditional database query, and the second conjunct *AlbumColor='red'* would be addressed to a subsystem such as QBIC. Thus, two different subsystems (in this case, perhaps a relational database management system to deal with the first conjunct, along with QBIC to deal with the second conjunct) would be involved in answering the query. Garlic has to piece together information from both subsystems in order to answer the query. Under the reasonable assumption that there are not many objects that satisfy the first conjunct *Artist='Beatles'*, a good way to evaluate this query would be to first determine all objects that satisfy the first conjunct (call this set of objects S), and then to obtain grades from QBIC (using random access) for the second conjunct for all objects in S. We can thereby obtain a grade for all objects for the full query. If the artist is not the Beatles, then the grade for the object is 0 (since the minimum of 0 and any grade is 0). If the artist is the Beatles, then the grade for the object is the grade obtained from QBIC in evaluating the second conjunct (since the minimum of 1 and any grade g is g). Note that, as we would expect, the result of the full query is a graded set where (a) the only objects whose grade is nonzero have the artist as the Beatles, and (b) among objects where the artist is the Beatles, those whose album cover are closest to red have the highest grades.

Let us now consider a more challenging example of a conjunction of atomic queries, where more than one conjunct is "nontraditional". An example would be the query

$$\textit{(Color='red')} \wedge \textit{(Shape='round')} \tag{3}$$

that we mentioned earlier. For the sake of this example, we assume that one subsystem deals with colors, and a completely different subsystem deals with shapes. Let A_1 denote the subquery *Color='red'*, and let A_2 denote the subquery *Shape='round'*. The grade of an object x under the query above is the minimum of the grade of x under the subquery A_1 from one subsystem and the grade of x under the subquery A_2 from the second subsystem. Therefore, Garlic must again combine results from two different subsystems. Assume that we are interested in obtaining the top k answers (such as $k = 10$). This means that we want to obtain k objects with the highest grades on this query (along with their grades). If there are ties, then we want to arbitrarily obtain k objects and their grades such that for each y among these k objects and each z not among these k objects, $\mu_Q(y) \geq \mu_Q(z)$ for this query Q. There is an obvious naive algorithm:

1. Have the subsystem dealing with color to output explicitly the graded set consisting of all pairs $(x, \mu_{A_1}(x))$ for every object x.

2. Have the subsystem dealing with shape to output explicitly the graded set consisting of all pairs $(x, \mu_{A_2}(x))$ for every object x.

3. Use this information to compute

$$\mu_{A_1 \wedge A_2}(x) = \min\{\mu_{A_1}(x), \mu_{A_2}(x)\}$$

for every object x. For the k objects x with the top grades $\mu_{A_1 \wedge A_2}(x)$, output the object along with its grade.

For this algorithm, the middleware cost is linear in the database size (the number of objects). Can we do any better?

Let us generalize beyond the query (3) above, which is the conjunction of two atomic queries, to consider conjunctions $A_1 \wedge \ldots \wedge A_m$ of m atomic queries. An important case arises when these conjuncts are independent (as they are, at least intuitively, in (3)). We shall be somewhat informal here. The full probabilistic model, including the definition of "independent", appears in [Fa96]. The next theorem shows that we can do substantially better than the naive algorithm.

Theorem 4.1: [Fa96] *There is an algorithm for finding the top k answers to each monotone query $F_t(A_1, \ldots, A_m)$, where $A_1, \ldots, A_m$ are independent, with middleware cost $O(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability, where N is the database size.*

In particular, this theorem applies to the conjunction $A_1 \wedge \ldots \wedge A_m$ of atomic queries, when the aggregation function is monotone. This includes any aggregation function obtained by iterating triangular norms (such as min), and in fact almost any reasonable choice for evaluating the conjunction. Note that in the case $m = 2$, which corresponds to the conjunction of two atomic queries, the cost is of the order of the square root of the size of the database. By "with arbitrarily high probability", we mean that for every $\epsilon > 0$, there is a constant c such that for every N, the probability that the middleware cost is more than $cN^{(m-1)/m}k^{1/m}$ is less than ϵ. It is shown in [Fa96] that for strict queries, there is a matching lower bound. That is, if $\mathcal{A}$ is an algorithm for finding the top k answers to a strict query $F_t(A_1, \ldots, A_m)$, where $A_1, \ldots, A_m$ are independent, then for every $\epsilon > 0$, there is a constant c' such that for every N, the probability that the middleware cost is less than $c'N^{(m-1)/m}k^{1/m}$ is less than ϵ. Thus, we have the following theorem, where as usual Θ means that is a matching upper and lower bound (up to a constant factor).

Theorem 4.2: [Fa96] *The middleware cost for finding the top k answers to a monotone, strict query $F_t(A_1, \ldots, A_m)$, where $A_1, \ldots, A_m$ are independent, is $\Theta(N^{(m-1)/m}k^{1/m})$, with arbitrarily high probability, where N is the database size.*

Intuitively, Theorem 4.2 tells us that we have matching upper and lower bounds for many natural notions of conjunction, such as all triangular norms.

We now give an algorithm that (as is shown in [Fa96]) fulfills the conditions of Theorem 4.1. This algorithm is called $\mathcal{A}_0$ in [Fa96]. It returns the top k answers for a

monotone query $F_t(A_1, \ldots, A_m)$, which we denote by Q. We assume that there are at least k objects, so that "the top k answers" makes sense. Assume that subsystem i evaluates the subquery A_i. We describe the algorithm informally; a more formal description appears in [Fa96].

The algorithm consists of three phases: sorted access, random access, and computation.

1. For each i, give subsystem i the query A_i under sorted access. Thus, subsystem i begins to output, one by one in sorted order based on grade, the graded set consisting of all pairs $(x, \mu_{A_i}(x))$, where as before x is an object and $\mu_{A_i}(x)$ is the grade of x under query A_i. Wait until the intersection of the m lists is of size at least k. That is, wait until there is a set L of at least k objects such that each subsystem has output all of the members of L.

2. For each object x that has been seen, do random access to each subsystem j to find $\mu_{A_j}(x)$.

3. Compute the grade $\mu_Q(x) = t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x))$ for each object x that has been seen. Let Y be a set containing the k objects that have been seen with highest grades (ties are broken arbitrarily). The output is then the graded set $\{(x, \mu_Q(x)) \mid x \in Y\}$.

Note that the algorithm has the nice feature that after finding the top k answers, in order to find the next k best answers we can "continue where we left off".

We now sketch a proof of the correctness of the algorithm; more details, including the performance analysis, appear in [Fa96]. Let y be an object that is not seen when the algorithm is run, that is, which is not output by any of the subsystems during sorted access. For each x in L (where as above, L is a set of at least k objects that has been output by all of the subsystems), and for each subsystem i, we know that $\mu_i(y) \leq \mu_i(x)$: this is because x was output under sorted access by subsystem i while y was not. So by monotonicity of t, we know that $\mu_Q(y) = t(\mu_{A_1}(y), \ldots, \mu_{A_m}(y)) \leq t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x))$ $= \mu_Q(x)$. So there are at least k objects in the output with grades at least as high as that of y.

Since algorithm $\mathcal{A}_0$ fulfills Theorem 4.1, it follows from Theorem 4.2 that algorithm $\mathcal{A}_0$ is optimal (up to a constant factor). In spite of this optimality, there are various improvements (as discussed in [Fa96]) that can be made to algorithm $\mathcal{A}_0$ (in particular, in the case when t is min, the standard aggregation function in fuzzy logic for the conjunction).

If the aggregation function t is not strict, then $\mathcal{A}_0$ is not necessarily optimal. An interesting example arises when t is max, which corresponds to the standard fuzzy disjunction $A_1 \vee \ldots \vee A_m$. In this case [Fa96], there is a simple algorithm whose middleware cost is only mk, independent of the size N of the database! Another example of a nonstrict

aggregation function is the median. It turns out [Fa96] that for the median over three attributes, just as in the case of max, there is an algorithm with a middleware cost that beats the lower bound of Theorem 4.2.

Chaudhuri and Gravano [CG96] consider ways to simulate algorithm $\mathcal{A}_0$ by using "filter conditions", which might say, for example, that the color score is at least .2.

4.2 Real-World Limitations on Algorithm $\mathcal{A}_0$

The algorithm $\mathcal{A}_0$ has been implemented in the Garlic system. Pritchard [Pr95] did an experimental implementation on an early version of Garlic, and found that although the algorithm is simple, there are many implementation issues that need to be addressed. Braendli did a more extensive implementation in a later version, as a part of an broad study carried out by Wimmers, Haas, Tork Roth, and Braendli [WHTB98], which considered implementation and performance issues. We discuss some of these issues in this section.

The performance of $\mathcal{A}_0$, as measured in both [Pr95] and [WHTB98], is consistent with the theoretical analysis. Furthermore, in [WHTB98], they say:

> We have seen that Fagin's algorithm behaves well for a broad range of queries, and a broad range of access costs.

The issues and concerns were the applicability of the algorithm in practice, as we now discuss.

One issue arises from the fact that the algorithm is guaranteed to be correct only when the aggregation function is monotone. The Garlic implementers considered two options: (1) they could provide a fixed set of legal (i.e., monotone) aggregation functions, such as min and average, and require the user to use an aggregation function from this set, or (2) they could allow the user to use an arbitrary, user-defined aggregation function. To give the system and the user maximum flexibility, they chose the second option. This makes it necessary for the system to somehow guarantee monotonicity.

Another issue was deciding what type of algorithm $\mathcal{A}_0$ is. The implementers struggled over the issue of whether $\mathcal{A}_0$ should be treated as a join or a sort (these are the choices the optimizer has). Ultimately, they decided to treat $\mathcal{A}_0$ as a join: this was in part a pragmatic decision, brought on by the fact that it was easier to teach the Garlic code about ordering requirements in the join phase rather than teaching the ordering code about multiple input streams. There are many issues brought on by the question of exactly how to implement algorithm $\mathcal{A}_0$. After all, we have described this algorithm only at a very high level, and many details need to be resolved.

Another issue arose from an important assumption underlying the algorithm $\mathcal{A}_0$: given an object from one input stream, the algorithm needs to be able to find the matching attributes of the same object in the second stream (this is what we referred to as "random access"). Since we are dealing with multiple subsystems, the "same" object might have

different identities in different subsystems. Even if there is some correspondence between object id's in different subsystems, Garlic has to be sure that the mapping is one-to-one.

Other problems are brought on by the fact that Garlic deals with complex objects. As an example, let us assume that the system contains information about Advertisements, which are complex objects with AdPhotos among their subobjects. Assume that we are interested in Advertisements with an AdPhoto that is red. Then Garlic would ask the subsystem that deals with AdPhotos to return all AdPhotos in sorted order, based on redness. To apply algorithm $\mathcal{A}_0$ to a conjunction of subqueries, where one conjunct asks for AdPhotos that are red, we need to be able to obtain object id's for Advertisements from the object id's of their AdPhotos. This information may not be easily available (e.g., through an index). Further, this is complicated by the fact that different multimedia objects can share the same component objects.

Finally, there are cost modeling issues. In order to use an optimizer, we need to understand the cost of applying various operators over various data in various repositories.

The reader is encouraged to read Wimmers et al.'s interesting paper [WHTB98], which gives much more detail, and discusses the solutions Garlic chose.

5 Weighting the Importance of Subqueries

In this section, we consider an interesting and important issue that arises for fuzzy queries in a multimedia database, but not for traditional queries in a standard database. As an example, let us consider again the query

$$(Color=\text{'red'}) \wedge (Shape=\text{'round'}) \tag{4}$$

What if the user cares twice as much about color as shape? Intuitively, we would then wish to assign twice as much weight to color as to shape. This would not make sense for traditional queries, but it does for fuzzy queries. In the user interface in a multimedia database, *sliders* are one mechanism for conveying to the system information about how much weight to assign to various attributes. Sliders are bars on the screen that indicate the importance of each attribute. The user moves his mouse to slide an indicator along the bar in order to increase or decrease the weighting of a given attribute.

How do we make sense of this weighting? That is, given an aggregation function f, what is the "weighted version" of f? There is one aggregation function where the answer is easy, namely, the average. For the weighted case, assume that the weight assigned to the color score in (4) is θ_1, and the weight assigned to the shape score is θ_2, where θ_1, θ_2 are nonnegative and sum to 1. Thus, if we care twice as much about the color as the shape, then we would take $\theta_1 = 2/3$ and $\theta_2 = 1/3$. When the aggregation function is the average, then we would "modify" it in the weighted case by taking the score assigned to (4) to be simply $\theta_1 x_1 + \theta_2 x_2$.

But what if the aggregation function is, say, the min? Then we cannot simply take the result to be $\theta_1 x_1 + \theta_2 x_2$. For example, if we are indifferent to color versus shape, so

that we weight them equally with $\theta_1 = \theta_2 = 1/2$, then we would get the wrong answer by using $\theta_1 x_1 + \theta_2 x_2$, since this does not give us the min of x_1 and x_2. (We are assuming here that we use the underlying, or "unweighted", rule for combining scores when the θ_i's are equal. Later, we shall make such assumptions explicit.) What should the answer be, as a function of x_1, x_2, θ_1, and θ_2?

Fagin and Wimmers [FW97] give an explicit formula for incorporating weights. Instead of restricting attention to aggregation functions, they consider a general context of "rules". A rule is simply an assignment of a value to every tuple, of varying sizes. For example, the rule corresponding to taking the average is to take the average of the entries in the tuple, and the rule corresponding to taking the min is to take the min of the entries in the tuple. The formula given in [FW97] is surprisingly simple, in that it involves far fewer terms than one might have guessed. It has three further desirable properties. The first desirable property is that when all of the weights are equal, then the result is obtained by simply using the underlying rule. Intuitively, this says that when all of the weights are equal, then this is the same as considering the unweighted case. The second desirable property is that if a particular argument has zero weight, then that argument can be dropped without affecting the value of the result. The third desirable property is that the value of the result is a continuous function of the weights. It is shown in [FW97] that if these three desirable properties hold, then under one additional assumption (a type of local linearity), their formula gives the unique possible answer.

We now describe the framework of [FW97], somewhat adapted to our application. Let us generalize beyond (4), by assuming that there are m conjuncts (not just two as in (4)), and that the score of conjunct i is x_i, where $0 \leq x_i \leq 1$. In (4), the color score would be x_1, where values of x_1 near 1 correspond to very red objects, and values near 0 correspond to objects that are very far from being red. Similarly, in (4), the shape score is x_2. Let f be a function whose domain is the set of all tuples (of all sizes) over $[0,1]$, and with range $[0,1]$. Then $f(x_1, \ldots, x_m)$ is the overall score. (Note that we are loosening up our definition of an aggregation function, by allowing tuples of arbitrary size as arguments.) We assume for notational convenience here that f is symmetric, but this is not really essential.[3]

Assume that $\theta_1, \ldots, \theta_m$ are all nonnegative and sum to one. Then we refer to $\Theta = (\theta_1, \ldots, \theta_m)$ as a *weighting*. Intuitively θ_i is the weight of attribute i. For each weighting $\Theta = (\theta_1, \ldots, \theta_m)$, we obtain (using the methodology of [FW97]) a function f_Θ whose domain consists of tuples of length m (the length of Θ). Intuitively, if $X = (x_1, \ldots, x_m)$, then $f_\Theta(X)$ is the overall score when the weights are given by the weighting Θ. If $\theta_1 \geq \ldots \geq \theta_m$, then we refer to the weighting $\Theta = (\theta_1, \ldots, \theta_m)$ as *ordered*. For notational simplicity, in this paper we restrict our attention to ordered weightings (although this restriction did not appear in [FW97]).

The following desiderata are given in [FW97] for the functions f_Θ:

[3]The paper [FW97] actually deals with families of functions, and the formal definition of "symmetric" is somewhat technical.

D1. $f_{(\frac{1}{m},\ldots,\frac{1}{m})}(x_1,\ldots,x_m) = f(x_1,\ldots,x_m)$. That is, if all of the weights in Θ are equal, then the "weighted" function f_Θ coincides with the "unweighted" function f.

D2. $f_{(\theta_1,\ldots,\theta_{m-1},0)}(x_1,\ldots,x_m) = f_{(\theta_1,\ldots,\theta_{m-1})}(x_1,\ldots,x_{m-1})$. That is, if a particular argument has zero weight, then that argument can be dropped without affecting the value of the result.

D3. $f_{(\theta_1,\ldots,\theta_m)}(x_1,\ldots,x_m)$ is a continuous function of $\theta_1,\ldots,\theta_m$.

The choice given in [FW97] for $f_{(\theta_1,\ldots,\theta_m)}(x_1,\ldots,x_m)$ is as follows (modified to fit our terminology), when $\theta_1 \geq \ldots \geq \theta_m$:

$$\begin{aligned} &(\theta_1 - \theta_2)\cdot f(x_1) + \\ &2\cdot(\theta_2 - \theta_3)\cdot f(x_1,x_2) + \\ &3\cdot(\theta_3 - \theta_4)\cdot f(x_1,x_2,x_3) + \\ &\cdots + \\ &m\cdot\theta_m\cdot f(x_1,\ldots,x_m). \end{aligned} \tag{5}$$

The formula in 5) is referred to as "the weighting formula". It is straightforward to verify that **D1**, **D2**, and **D3** are satisfied when we use the weighting formula for $f_{(\theta_1,\ldots,\theta_m)}(x_1,\ldots,x_m)$. It is also shown in [FW97] that the weighting formula is well-defined, even when some of the θ_i's are equal. For example, if $\theta_2 = \theta_3$, then we could reverse the roles of θ_2 and θ_3. Then the second summand in the weighting formula would involve $f(x_1,x_3)$ rather than $f(x_1,x_2)$. The point is that this does not matter, because even though $f(x_1,x_2)$ and $f(x_1,x_3)$ may be different, they are multiplied by $\theta_2 - \theta_3$, which is 0.

Although the weighting formula may look somewhat arbitrary, it is shown in [FW97] that it is actually uniquely determined, under one additional assumption that we now discuss. We say that our collection of weighted functions f_Θ is *locally linear* if

$$f_{\alpha\cdot\Theta+(1-\alpha)\cdot\Theta'}(X) = \alpha\cdot f_\Theta(X) + (1-\alpha)\cdot f_{\Theta'}(X), \tag{6}$$

whenever Θ, Θ', X are of the same length, and $\alpha \in [0,1]$. (The definition of local linearity in [FW97] says essentially that (6) must hold whenever Θ and Θ' are ordered. For us, this is always the case, since we define f_Θ only when Θ is ordered.) Note that $\alpha\cdot\Theta+(1-\alpha)\cdot\Theta'$ is ordered whenever Θ and Θ' are ordered.

Define the condition **D3′** as follows:

D3′. The collection of weighted functions f_Θ is locally linear.

Condition **D3′** implies condition **D3** above (that $f_\Theta(X)$ is a continuous function of $\theta_1,\ldots,\theta_m$) [FW97]. Furthermore, the choice of the weighting formula for $f_\Theta(X)$ is the unique one that satisfies **D1**, **D2**, and **D3′** [FW97].

We now discuss local linearity. Intuitively, local linearity says that the aggregation functions act like a balance. Local linearity demands that the weighting that is the midpoint[4] of two ordered weightings should produce a value that is the midpoint of the two values produced by the given weightings, or in other terms that (6) must hold when Θ and Θ' are ordered, and so agree on which search term is the most important, which is the second most important, etc. Local linearity says that in this case, we do a linear interpolation, which is a very natural assumption. Another argument in favor of local linearity is that it leads to such a nice weighting formula.

The weighting formula is a convex combination of the values $f(x_1)$, $f(x_1, x_2)$, $f(x_1, x_2, x_3)$, ..., $f(x_1, \ldots, x_m)$, since the coefficients $(\theta_1 - \theta_2)$, $2 \cdot (\theta_2 - \theta_3)$, $3 \cdot (\theta_3 - \theta_4)$, ..., $m \cdot \theta_m$ are nonnegative and sum to 1. This is quite useful: for example, it follows that the range of the weighted rule represented by the weighting formula is $[0, 1]$, since the range of the unweighted rule f is $[0, 1]$. A surprising feature of the weighting formula is that it depends only on the m terms $f(x_1)$, $f(x_1, x_2)$, $f(x_1, x_2, x_3)$, ..., $f(x_1, \ldots, x_m)$, and not on any of the other possible terms, such as $f(x_2)$, $f(x_1, x_3)$, and so on. *A priori*, we might have believed that $f_\Theta(X)$ would depend on all of the $2^m - 1$ such terms.

It is shown in [FW97] that monotonicity and strictness of the (unweighted) f is inherited by the (weighted) functions f_Θ. In particular, algorithm $\mathcal{A}_0$ continues to be correct and optimal in the weighted case, where of course the "final score" is obtained by using f_Θ rather than f.

6 Summary and Open Problems

Multimedia databases have interesting new issues beyond those of traditional databases. In this paper, we discussed some of these issues involving multimedia queries. These include:

- *The complexity of resolving atomic queries.* Here there are two issues. First, it is often computationally expensive to decide if a given object satisfies an atomic query. Second, there is typically a large middleware cost in finding those objects that satisfy the query. In particular, we wish to avoid doing a sequential scan of the entire database, and instead use, for example, an indexing technique. Developing techniques to speed up the evaluation of atomic multimedia queries is an important problem, that requires more research.

[4] In fact, local linearity extends beyond the midpoint to any weighting that is a convex combination of two ordered weightings: if a weighting is a convex combination of two ordered weightings, then local linearity demands that the associated value should be the same convex combination of the values associated with the given weightings. In this paper, our life is made simpler by the fact that we consider weighted functions f_Θ only for weightings Θ that are ordered. In [FW97], where the function f_Θ is defined even when Θ is not ordered, it is shown that an assumption that (6) holds for *every* choice of Θ and Θ' would be incompatible with the properties **D1** and **D2**, unless $f(x_1, \ldots, x_k) = \frac{f(x_1) + \cdots + f(x_k)}{k}$ for every k.

- *Handling Boolean combinations of atomic queries.* The author has made a first step, by giving a reasonable semantics, involving aggregation functions, for evaluating Boolean combinations, and by giving an efficient algorithm for taking conjunctions of atomic queries, that is optimal under certain natural assumptions. In particular, for the case of two independent conjuncts, the middleware cost of the algorithm is of the order of the square root of the database size. Even here, as discussed in [WHTB98], there are numerous practical issues as to when the algorithm can be applied. Furthermore, it is hopeless to find efficient algorithms in general: in particular, in [Fa96] the author gives a (somewhat artificial) case where the middleware cost is necessarily linear in the database size (in other words, there is a provable linear lower bound). Finding efficient algorithms in various natural cases that arise in practice is an interesting open problem. In fact, at this point, due to our lack of experience, it is not even clear what queries will arise in practice.

- *Restricted modes of access.* One of the interesting differences that arises in the multimedia case is that there are often restricted modes of access to a multimedia repository (such as what we call in this paper sorted access and random access). This leads to another problem: to give a more realistic cost measure than the definition in [Fa96] for the middleware cost. This is especially important in the presence of query optimizers.

- *Weighting the importance of subqueries.* This is a good example of an issue that arises for multimedia databases but not for traditional databases. Here we feel comfortable with the solution from [FW97], which gives a general formula that tells how to modify a rule to apply to the case where weights are assigned to the importance of each argument. In our case, we use this formula to weight the importance of the various conjuncts in a conjunction.

- *Syntax and user interface.* We have been fairly simplistic in this paper as to the syntactical form of queries. In an actual system, the queries would have a much more complicated syntax. They could possibly be written in an SQL-like form [CB74, DD94], as is done in [WHTB98], or they could be given more graphically, as is done in the QBIC system. Getting the right user interface is a challenging problem.

Multimedia databases are an increasingly important area, which involve fascinating new issues beyond those of traditional databases. A great deal of new research is required.

Acknowledgments

The author is grateful to Laura Haas, Jim Hafner, Wayne Niblack, Dragutin Petkovic, and Ed Wimmers for helpful comments and suggestions.

but no clearcut "best" choice for combining

References

[ACH+90] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell, An Efficiently Computable Metric for Comparing Polygonal Shapes, *Proc. First ACM-SIAM Symposium on Discrete Algorithms* (1990), pp. 129–137.

[Al85] C. Alsina, On a Family of Connectives for Fuzzy Sets, *Fuzzy Sets and Systems* **16** (1985), pp. 231–235.

[BKSS90] N. Beckman, H.-P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, *Proc. ACM SIGMOD Conference* (1990), pp. 322–331.

[BD86] P. P. Bonissone and K. S. Decker, Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-Off Precision and Complexity, in *Uncertainty in Artificial Intelligence* (L. N. Kanal and J. F. Lemmer, Eds.), Amsterdam (1986)

[BG73] R. Bellman and M. Giertz, On the Analytic Formalism of the Theory of Fuzzy Sets, *Information Sciences* **5** (1973), pp. 149–156.

[CHS+95] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers, Towards Heterogeneous Multimedia Information Systems: The Garlic Approach, *RIDE-DOM '95 (5th Int'l Workshop on Research Issues in Data Engineering: Distributed Object Management)*, 1995, pp. 124-131.

[CB74] D. D. Chamberlin and R. F. Boyce, SEQUEL: A Structured English Query Language, *Proc. ACM SIGFIDET Workshop on Data Description Access, and Control* (1974), pp. 249–264.

[CG96] S. Chaudhuri and L. Gravano, Optimizing Queries over Multimedia Repositories, *Proc. ACM SIGMOD Conference* (1996), pp. 91–102.

[CHN+95] W. F. Cody, L. M. Haas, W. Niblack, M. Arya, M. J. Carey, R. Fagin, M. Flickner, D. S. Lee, D. Petkovic, P. M. Schwarz, J. Thomas, M. Tork Roth, J. H. Williams, and E. L. Wimmers, Querying Multimedia Data from Multiple Repositories by Content: The Garlic Project, *IFIP 2.6 3rd Working Conference on Visual Database Systems (VDB-3)*, 1995.

[DD94] C. J. Date with H. Darwen, *A Guide to the SQL Standard*, Third Edition, Addison-Wesley, New York, 1994.

[DP80] D. Dubois and H. Prade, *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York, 1980.

[DP84] D. Dubois and H. Prade, Criteria Aggregation and Ranking of Alternatives in the Framework of Fuzzy Set Theory, in *Fuzzy Sets and Decision Analysis* (H. J. Zimmermann, L. A. Zadeh, and B. Gaines, Eds.), TIMS Studies in Management Sciences **20** (1984), pp. 209–240.

[DP85] D. Dubois and H. Prade, A Review of Fuzzy Set Aggregation Connectives, *Information Sciences* **36** (1985), pp. 85–121.

[Fa96] R. Fagin, Combining Fuzzy Information from Multiple Systems, *J. Computer and System Sciences*, to appear. (Special issue for selected papers from the 1996 Symposium on Principles of Database Systems.) Preliminary version appeared in *Proc. Fifteenth ACM Symp. on Principles of Database Systems*, Montreal (1996), pp. 216–226.

[FW97] R. Fagin and E. Wimmers, Incorporating User Preferences in Multimedia Queries, Proc. 6th International Conference on Database Theory (1997), *Springer-Verlag Lecture Notes in Computer Science* **1186**, ed. F. Afrati and Ph. Kolaitis, pp. 247–261. A revised and expanded version, called *A Formula for Incorporating Weights into Scoring Rules*, is currently available from the author's home page, `http://www.almaden.ibm.com/cs/people/fagin/`

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz, Efficient and Effective Querying by Image Content, *J. of Intelligent Information Systems* **3** (1994), pp. 231–262.

[Gr97] W. I. Grosky, Managing Multimedia Information in Database Systems, *Comm. ACM* **40**, 12 (Dec. 1997), pp. 73–80.

[HSE+95] J. Hafner, H. S. Sawhney, W. Equitz, M. Flickner, and W. Niblack, Efficient Color Histogram Indexing for Quadratic Form Distance Functions, *IEEE Trans. Pattern Analysis and Machine Intelligence* **17**, 7 (July 1995), pp. 729–736.

[HRK92] D. Huttenlocher, W. Rucklidge, and G. Klanderman, Comparing Images Using the Hausdorff Distance under Translation, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (1992), pp. 654–656.

[Io89] M. Ioka, *A Method for Defining the Similarity of Images on the Basis of Color Information*, Technical Report RT-0030, IBM Tokyo Research Lab (1989).

[Ja89] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall (1989).

[KK97] Y-S. Kim and W-Y. Kim, Content-Based Trademark Retrieval System Using Visually Salient Feature, *IEEE Conf. on Computer Vision and Pattern Recognition*, Puerto Rico, 1997.

[MKC+91] R. McConnell, R. Kwok, J. C. Curlander, W. Kober, S. S. Pang, Ψ–S Correlation and Dynamic Time Warping: Two Methods for Tracking Ice Floes in SAR Images, *IEEE Trans. Geoscience and Remote Sensing* **29** (1991), pp. 1004–1012.

[MKL97] B. M. Mehtre, M. S. Kankanhalli, and W. F. Lee, Shape Measures for Content Based Image Retrieval: A Comparison, *Information Processing and Management* **33**, 3 (1997), pp. 319–337.

[Mu91] D. Mumford, Mathematical Theories of Shape: Do They Model Perception?, in: *Geometric Methods in Computer Vision*, volume 1570 (1991), SPIE, pp. 2–10.

[Mi89] M. Mizumoto, Pictorial Representations of Fuzzy Connectives, Part I: Cases of T-norms, T-conorms and Averaging Operators, *Fuzzy Sets and Systems* **31** (1989), pp. 217–242.

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker, The QBIC Project: Querying Images by Content Using Color, Texture and Shape, *SPIE Conference on Storage and Retrieval for Image and Video Databases* (1993), volume 1908, pp. 173–187. QBIC Web server is `http://wwwqbic.almaden.ibm.com/`

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, The Grid File: An Adaptable, Symmetric Multikey File Structure, *ACM Trans. on Database Systems* **9**, 1 (1984), pp. 38–71.

[Ot92] M. Otterman, *Approximate Matching with High Dimensionality R-Trees*, M.Sc. Scholarly paper, Dept. of Computer Science, Univ. of Maryland (1992).

[Pr95] A. Pritchard, *Combining Graded Query Results in the Garlic Multimedia Information System*, unpublished manuscript (1995).

[Sa89] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison Wesley (1989).

[SC96] J. R. Smith and S-F. Chang, *Searching for Images and Videos on the World-Wide Web*, Technical Report #459-96-25, Center for Telecommunications Research, Columbia University (1996).

[SS63] B. Schweizer and A. Sklar, Associative Functions and Abstract Semi-groups, *Publ. Math. Debrecen* **10** (1963), pp. 69–81.

[SO95] M. Stricker and M. Orengo, Similarity of Color Images, *SPIE Conference on Storage and Retrieval for Image and Video Databases III* (1995), volume 2420, pp. 381–392.

[TC91] G. Taubin, G. and D. B. Cooper, Recognition and Positioning of Rigid Objects Using Algebraic Moment Invariants, in: *Geometric Methods in Computer Vision*, volume 1570 (1991), SPIE, pp. 175–186.

[TZZ79] U. Thole, H.-J. Zimmermann, and P. Zysno, On the Suitability of Minimum and Product Operators for the Intersection of Fuzzy Sets, *Fuzzy Sets and Systems* **2** (1979), pp. 167–180.

[VG83] W. Voxman and R. Goetschel, A Note on the Characterization of the Max and Min Operators, *Information Sciences* **30** (1983), pp. 5–10.

[Wi98] E. L. Wimmers, Minimal Bellman-Giertz Theorems, to appear.

[WHTB98] E. L. Wimmers, L. M. Haas, M. Tork Roth, and C. Braendli, *Using Fagin's Algorithm for Merging Ranked Results in Multimedia Middleware*, to appear (1998).

[Ya82] R. R. Yager, Some Procedures for Selecting Fuzzy Set-Theoretic Operations, *International Journal General Systems* **8** (1982), pp. 115-124.

[Za65] L. A. Zadeh, Fuzzy sets, *Information and Control* **8** (1965), pp. 338–353.

[Zi96] H.-J. Zimmermann, *Fuzzy Set Theory*, Kluwer Academic Publishers, Boston (1996).