# Fuzzy queries in multimedia database systems

Ron Fagin

IBM Almaden Research Center
San Jose, California

# Multimedia versus Relational Databases

A relational database consists of a set of tables.

| *ARTIST* | *ALBUM* | *YEAR* |
|---|---|---|
| Beatles | Abbey Road | 1969 |
| Beatles | Let It Be | 1970 |
| Michael Jackson | Thriller | 1982 |
| ... | ... | ... |

The result of a query is a relation.

- Possibly a unary relation, i.e. a set.

Give names of all albums where artist=Beatles

**Result:** set of names of albums

Entries in multimedia database may be much more complicated than entry in a relational database.

- Multimedia database might deal with pictures with complicated coloring pattern and with a number of shapes.

Multimedia queries are different from traditional relational queries.

- Might ask for images most similar to a given image.

- No clearcut set of “right answers”.

These differences lead to interesting new issues.

# Atomic Queries

**Example of atomic multimedia query:**

Give all album covers with this shade of red

**Result:** sorted list of album covers

Need notion of closeness of color.

Ioka's method [1989], used in QBIC:

Each object has $k$-element color histogram.

- Typical values of $k$ are 64, 100, or 256.

Let $\mathbf{x}$ and $\mathbf{y}$ be two $k$-dimensional vectors representing color histograms of two objects.

Color distance between the objects taken to be:

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

- $A_{ij}$ describes similarity between color $i$ and color $j$.

# The Complexity of Resolving Atomic Queries

Complex in two senses:

- computationally expensive to decide if a given object satisfies an atomic query.
  - **Example:** Must compute

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

- large database access cost in finding those objects that satisfy the query.
  - In particular, want to avoid doing a sequential scan of the entire database.

# Speeding Up the Evaluation

**Natural thing to try:** multidimensional indexing method

**Problem:** Dimensionality curse

- grid files [Nievergelt, Hinterberger, and Sevcik 1984]
  - grow exponentially with dimensionality
- linear quadtrees [Samet 1989]
  - grow exponentially with dimensionality
- R*-trees [Beckman, Kriegel, Schneider, and Seeger 1990]
  - Robust for dimensions up to around 20 [Otterman 1992]

Much more work needed in high-dimensional indexing, or similar techniques, to efficiently evaluate atomic multimedia queries.

# Distance-Bounding Strategy

[Hafner, Sawhney, Equitz, Flickner, and Niblack 1995]

$d(\mathbf{x}, \mathbf{y})$ is distance between color histograms $\mathbf{x}$ and $\mathbf{y}$.

Associate with (long) color feature vector $\mathbf{x}$ a short (say, dimension 3) color vector $\hat{\mathbf{x}}$ that "summarizes" $\mathbf{x}$.

Give simple-to-compute distance measure $\hat{d}$ between small color vectors.

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

Want to find objects whose color feature vector $\mathbf{y}$ is close to target color feature vector $\mathbf{x}$.

Use $\hat{d}$ as 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.

- Can have multidimensional index on short color vectors.

- Even without an index, often need only compute $\hat{d}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ rather than $d(\mathbf{x}, \mathbf{y})$.

# Boolean Combinations of Atomic Queries

**Example of relational query:**

Give names of all albums where artist=Beatles

**Result:** set of names of albums

**Example of multimedia query:**

Give all album covers with this shade of red

**Result:** sorted list of album covers

Difference in semantics: set versus sorted list

- If we try to combine relational and multimedia queries, this difference causes a mismatch.

# Combining These

How could we combine relational queries and multimedia queries?

**Example:**

> Give names of all albums where artist=Beatles and where the album cover has this shade of red

In this case, probably want sorted list, where

- the list contains only albums by the Beatles, and
- the list is sorted by goodness of match in color.

*What about more complicated queries, such as Boolean combinations?*

**Example** (replacing "and" above by "or"):

> Give names of all albums where artist=Beatles or where the album cover has this shade of red

*Is the answer a set, a sorted list, or some combination?*

Could even combine multimedia queries:

$$(\text{Color} = \text{red}) \wedge (\text{Shape} = \bigcirc)$$

Is the answer a sorted list? If so, what is the ordering?

# The Solution: Graded Sets

- *Atomic queries*: of the form $X = t$, where $X$ is the name of an attribute, and $t$ is a target.

  **Examples:**

  Artist = Beatles
  AlbumColor = red

- For each atomic query $Q$, grade $\mu_Q(x)$ between 0 and 1 assigned to each object $x$
  - Grade tells how well object $x$ fulfills atomic query $Q$ (perfect match = 1).
  - For relational queries, such as (Artist = Beatles), grade is either 0 (false) or 1 (true).
  - Multimedia queries, such as (AlbumColor = red), may be assigned grade strictly between 0 and 1.

- Graded sets generalize both sets and sorted lists
  - Convenient to think of graded set as corresponding to sorted list
    - User might be presented first with, say, top 10 objects.

# Semantics of $m$-ary Queries

An *$m$-ary combining function* is a function $t$ from $[0,1]^m$ to $[0,1]$.

Define the *$m$-ary query $Q$ based on $t$* via

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

What combining functions $t$ should we use for conjunction, disjunction, and negation?

A desirable property for conjunction and disjunction:

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

A desirable property for conjunction:

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

A query is *monotonic* if it is based on a monotonic combining function $t$; similarly for *strict*.

# Standard Rules of Fuzzy Logic

- $\mu_{A \wedge B}(x) = \min\{\mu_A(x), \mu_B(x)\}$
- $\mu_{A \vee B}(x) = \max\{\mu_A(x), \mu_B(x)\}$
- $\mu_{\neg A}(x) = 1 - \mu_A(x)$

Attractive for several reasons:

- $\wedge$ and $\vee$ are monotonic, and $\wedge$ is strict.
- Conservative extension of database semantics.
  - For relational queries, object assigned grade 1 iff part of standard answer
- Logically equivalent queries involving only conjunction and disjunction assigned the same grade.
  - $\mu_{A \wedge A}(x) = \mu_A(x)$
  - $\mu_{A \wedge (B \vee C)}(x) = \mu_{(A \wedge B) \vee (A \wedge C)}(x)$
  - Optimizer can replace query by logically equivalent query

# Min and Max are Unique Choice

**Theorem** [Bellman and Giertz 1973; Yager 1982; Voxman and Goetschel 1983; Dubois & Prade 1984; Wimmers 1998]. The unique choices for evaluating $\wedge$ and $\vee$ that are monotonic and that preserve logical equivalence are min and max.

# Query Evaluation

Now have semantics for $m$-ary queries.

How do we evaluate them?

# Middleware

Because of many varieties of data a multimedia database system must handle, such a system is often really middleware.

- The system is on top of various subsystems, and integrates results from the subsystems.

- **Example:** Garlic

Given the query:

$$(\text{Artist} = \text{Beatles}) \wedge (\text{AlbumColor} = \text{red})$$

Garlic might depend on DB2 to deal with the subquery $\text{Artist} = \text{Beatles}$, and QBIC to deal with the subquery $\text{AlbumColor} = \text{red}$.

- Garlic receives and combines the information.

# Interface to QBIC

What interface can we assume between multimedia system like Garlic, and subsystem like QBIC?

- The subsystem can output graded sets sorted by grade ("sorted access").

- The subsystem can give the grade of an object upon request ("random access").

Middleware cost: $c_1 \cdot$(number of elements requested in sorted access) + $c_2 \cdot$(number requested in random access).

- Neglecting cost within the black box, which can vary greatly, and which we don't know how to model.

# Algorithms for Query Evaluation

Pure multimedia query:

$$(\text{Color} = \text{red}) \wedge (\text{Shape} = \bigcirc)$$

How to obtain top $k$ answers?

Naive algorithm:

1. Obtain the redness score of every object in the database.
2. Obtain the roundness score of every object in the database.
3. Calculate the overall score of every object.
4. Select the top $k$ answers.

This algorithm has middleware cost linear in the database size.

Can we do better?

# Upper Bound

Probability model: Each permutation of the objects for each attribute is equally likely, independent of the permutations for the other attributes.

- "The attributes are independent".

**Theorem.** There is an algorithm for finding the top $k$ answers to each monotone query on $m$ attributes, with middleware cost $O(N^{(m-1)/m}k^{1/m})$ with arbitrarily high probability, where $N$ is the database size.

- If $m = 1$, cost is $O(k)$.

- If $m = 2$, cost is $O(\sqrt{Nk})$.

# Lower bound

**Theorem.** For strict queries, there is a matching lower bound.

- median (on 3 attributes) and max are not strict, and can be evaluated with middleware cost that beats the lower bound.

# Algorithm

1. Sorted access phase: Do sorted access for each $A_i$ until intersection of lists of size at least $k$ (so some set of $k$ objects is in all of the lists).

2. Random access phase: For each object $x$ that has been seen, do random access for each $A_i$ to find $\mu_{A_i}(x)$.

3. Computation phase: Compute the grade $\mu_Q(x) = t(\mu_{A_1}(x), \ldots, \mu_{A_m}(x))$ for each object $x$ that has been seen. The winners are the $k$ objects with the highest overall grades $\mu_Q(x)$.

Features of the algorithm:

- Always gives the right answer.
- With high probability, the middleware cost is small.
- Cumulative: can get next $k$ answers by continuing where the algorithm left off.

# The Algorithm in Action

Sorted access phase:

| | | | |
|---|---|---|---|
| 371: | .987 | 2509: | .993 |
| 25: | .984 | 129: | .985 |
| 257: | .979 | 426: | .963 |
| 513: | .972 | 9: | .951 |
| 9112: | .953 | 4387: | .939 |
| 2614: | .871 | 257: | .936 |
| 8005: | .863 | 107: | .927 |
| 129: | .755 | 604: | .908 |

Random access phase:

| | | | |
|---|---|---|---|
| 2509: | .413 | 371: | .209 |
| 426: | .561 | 25: | .463 |
| 9: | .029 | 513: | .846 |
| 4387: | .393 | 9112: | .097 |
| 107: | .702 | 2614: | .516 |
| 604: | .633 | 8005: | .802 |

Computation phase winners:

| | |
|---|---|
| 257: | .936 |
| 513: | .846 |

# Implementation of the Algorithm

Implemented in Garlic by

- Pritchard (early version of Garlic)
- Braendli

Performance and implementation issues addressed in [Wimmers, Haas, Tork Roth, and Braendli 1998].

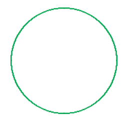
Performance is stopwatch time, so is a measure of total system performance.

They conclude that the algorithm

> "behaves well for a broad range of queries, and a broad range of access costs."

# Weighting the Importance of Subqueries

Consider query

$(\text{Color} = \text{red}) \wedge (\text{Shape} =$  $)$

What if the user cares twice as much about color as shape?

- Can represent with sliders

Assume a 2-ary combining function. Assume weight of first argument is $\theta_1$, and of second argument is $\theta_2$ (with $\theta_1 + \theta_2 = 1$).

- If combining function is average $(a_1 + a_2)/2$, then weighted combining function is weighted average $\theta_1 a_1 + \theta_2 a_2$.

- What about other situations (such as when combining function is min)?

  - If combining function is min, can't use $\theta_1 a_1 + \theta_2 a_2$ in weighted case, since get wrong answer if $\theta_1 = \theta_2 = \frac{1}{2}$.

# Our Solution

Joint work with Ed Wimmers.

Generalize beyond combining functions to "scoring rules".

- A *scoring rule* is an assignment of a value to every tuple (of varying sizes).
- Entries of tuples need not be numeric.

# Desirable Properties

*Based on unweighted rule:*

- $f_{(\frac{1}{3},\frac{1}{3},\frac{1}{3})}(x_1, x_2, x_3) = f(x_1, x_2, x_3)$

*Compatibility:*

- $f_{(0,\theta_2,\theta_3))}(x_1, x_2, x_3) = f_{(\theta_2,\theta_3)}(x_2, x_3)$
- $f_{(\theta_1,0,\theta_3))}(x_1, x_2, x_3) = f_{(\theta_1,\theta_3)}(x_1, x_3)$
- $f_{(\theta_1,\theta_2,0))}(x_1, x_2, x_3) = f_{(\theta_1,\theta_2)}(x_1, x_2)$

*Continuity in the $\theta_i$'s*

# Linearity

Let

$\Theta = (\theta_1, \ldots, \theta_m)$

$\Theta' = (\theta'_1, \ldots, \theta'_m)$

Would like

*Linearity:*

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

But:

**Theorem:** There is no extension to the weighted case that fulfills all four desirable properties, unless the unweighted rule $f$ satisfies

$$f(x_1, \ldots, x_m) = \frac{f(x_1) + \cdots + f(x_m)}{m}$$

for every $m$.

So we must settle for something less than linearity.

# Local Linearity

Two weightings are *order-equivalent* if they agree on the order of importance of the arguments.

**Example:**

- $(.3, .5, .2)$ and $(.2, .7, .1)$ are order-equivalent.

*Local linearity* demands linearity only among order-equivalent weightings.

**Example:**

- $(.3, .5, .2)$ and $(.2, .7, .1)$ are order-equivalent.
- $(.26, .58, .16) = \frac{3}{5} \cdot (.3, .5, .2) + \frac{2}{5} \cdot (2., .7, .1)$
- $f_{(.26,.58,.16)} = \frac{3}{5} \cdot f_{(.3,.5,.2)} + \frac{2}{5} \cdot f_{(2.,.7,.1)}$

**Theorem:** Local linearity implies continuity in the $\theta_i$'s.

# Our Goal

Our goal:

- Obtain a procedure for extending to the weighted case that is
  - based on unweighted rule,
  - compatible, and
  - continuous in the $\theta_i$'s.

We do even better:

- Our procedure is
  - based on unweighted rule,
  - compatible, and
  - locally linear.

- Instead of just a procedure, we get a simple closed-form formula.

# Weighting Formula

Assume $\theta_1 \geq \cdots \geq \theta_m$, and $\theta_1 + \ldots + \theta_m = 1$.

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

- Involves $m$ terms.
- Does not need, for example, $f(x_2, x_3)$, $f(x_1, x_4, x_7)$.
- Formula well defined (even when some $\theta_i$'s equal).
- This is a weighted linear combination:

$$
\begin{aligned}
(\theta_1 - \theta_2) + 2 \cdot (\theta_2 - \theta_3) + 3 \cdot (\theta_3 - \theta_4) + \ldots + m \cdot \theta_m \\
= \theta_1 + \ldots + \theta_m = 1
\end{aligned}
$$

**Theorem:** For every unweighted rule, our formula gives the unique extension to a weighted rule that is based on unweighted rule, compatible, and locally linear.

Our formula

- has all three desirable properties
- is the only formula having these three desirable properties
- is surprisingly simple: only $m$ terms (not $2^m$)
- preserves properties such as monotonicity and strictness
- always works

# Average

Say $m = 3$. The formula gives

$$(\theta_1 - \theta_2)x_1 + 2(\theta_2 - \theta_3)\frac{x_1 + x_2}{2} + 3\theta_3\frac{x_1 + x_2 + x_3}{3}$$
$$= \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3.$$

This is the right answer.

# Min

If 3 arguments, and $\theta_1 \geq \theta_2 \geq \theta_3$:

$$(\theta_1 - \theta_2)x_1 + 2(\theta_2 - \theta_3)\min(x_1, x_2) + 3\theta_3 \min(x_1, x_2, x_3).$$

Note how simple this formula is.

## Allowing Arbitrary Arguments in Tuples

No reason that arguments must be numerical scores.

General formulation:

- Given an arbitrary scoring rule for assigning values to tuples, our formula assigns values to tuples where we can weight the importance of the entries.

# Information Retrieval

**Example:** using a search engine on the Web

- Arguments are search terms.
- Our formula allows us to weight importance of each search term.
- In most search engines, this can be done with essentially no loss in efficiency over unweighted searches.
  - Search engine checks same entries in index.

Joint work with Yoëlle Maarek

# Summary

- The complexity of resolving atomic queries
- Handling Boolean combinations of atomic queries
- Restricted modes of access
- Weighting the importance of subqueries

# The Complexity of Resolving Atomic Queries

Scalability is major issue.

- Want to avoid doing a sequential scan of the entire database.

**Open problem:** Obtain techniques to greatly speed up the evaluation of atomic multimedia queries.

# Handling Boolean Combinations of Atomic Queries

- Have reasonable semantics, involving combining functions, for evaluating Boolean combinations

- Have efficient algorithm for taking conjunctions of atomic queries, that is optimal under certain natural assumptions.
  - For two independent conjuncts, middleware cost of the algorithm is $O(\sqrt{N})$.

- Can't get efficient algorithms in general.
  - Have query with provable linear lower bound on middleware cost.

- **Open problem:** Find efficient algorithms in various natural cases that arise in practice.
  - **Open problem:** What queries will arise in practice?

# Restricted Modes of Access

- One interesting differences for the multimedia case is that there are often restricted modes of access to a multimedia repository.
  - sorted access
  - random access
- **Open problem:** Obtain cost model for overall system performance.
  - Especially important in the presence of query optimizers.
  - **Open problem:** Is the algorithm optimal under these more comprehensive cost measures? If not, find algorithms that are.

# Weighting the Importance of Subqueries

- Issue arises for multimedia databases but not for relational databases.

- Have general formula that tells how to modify a scoring rule to apply to the case where weights are assigned to the importance of each argument.
  - Here we use this formula to weight the importance of the various conjuncts in a conjunction.

# Overall Summary

Multimedia databases are an increasingly important area, which involve fascinating new issues beyond those of traditional relational databases.

A great deal of new research is required.

# Papers


- R. Fagin, *Fuzzy queries in multimedia database systems*. *PODS '98*, pp. 1–10.

- R. Fagin, *Combining fuzzy information from multiple systems*. *J. Computer and System Sciences* 58, 1999, pp. 83–99.

- R. Fagin and E. L. Wimmers, *A formula for incorporating weights into scoring rules*. *Theoretical Computer Science*, to appear.

- R. Fagin and Y. Maarek, *Allowing users to weight search terms in information retrieval*. IBM Research Report, March 1998.

- E. L. Wimmers, L. M. Haas, M. Tork Roth, and C. Braendli, *Using Fagin's algorithm for merging ranked results in multimedia middleware*. *IFCIS Int'l Conf. on Cooperative Information Systems*, 1999.


**First four papers obtainable from my home page:**

`http://www.almaden.ibm.com/cs/people/fagin/`