Optimizing Multi-Feature Queries for Image Databases

Ulrich Güntzer
University of Tübingen
guentzer@informatik.uni-tuebingen.de

Wolf-Tilo Balke
University of Augsburg
balke@informatik.uni-augsburg.de

Werner Kießling
University of Augsburg
kiessling@informatik.uni-augsburg.de

Abstract

In digital libraries image retrieval queries can be based on the similarity of objects, using several feature attributes like shape, texture, color or text. Such multi-feature queries return a ranked result set instead of exact matches. Besides, the user wants to see only the k top-ranked objects. We present a new algorithm called Quick-Combine (European patent pending, nr. EP 00102651.7) for combining multi-feature result lists, guaranteeing the correct retrieval of the k top-ranked results. For score aggregation virtually any combining function can be used, including weighted queries. Compared to Fagin's algorithm we have developed an improved termination condition in tuned combination with a heuristic control flow adopting itself narrowly to the particular score distribution. Top-ranked results can be computed and output incrementally. We show that we can dramatically improve performance, in particular for non-uniform score distributions. Benchmarks on practical data indicate efficiency gains by a factor of 30. For very skewed data observed speed-up factors are even larger. These performance results scale through different database sizes and numbers of result sets to combine.

1 Introduction

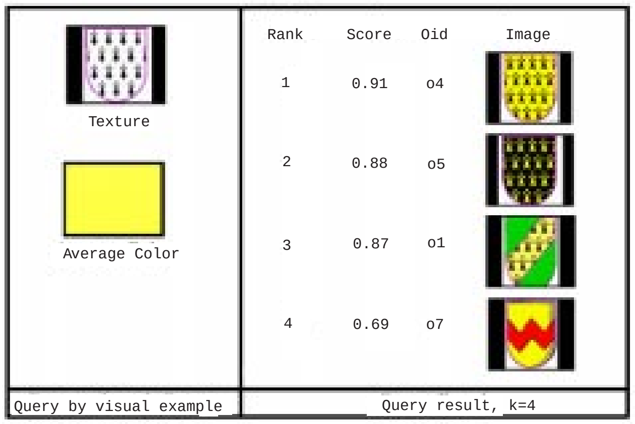
In universal database systems the handling of multimedia data such as images, video or audio files poses an increasingly demanding problem. The query evaluation model typically does not retrieve a set of exact matches but rather a ranked result set, where an aggregated score is attached to each object returned. Only a few top-ranked results are normally of interest to the user. Imagine a traditional image archive where every image is labeled with captions like name, registration number and related information like the photographer's name or the image's size. To retrieve images this meta-information has to be known for each search. In times of digitization, these archives are increasingly replaced by modern image databases, but most systems still only focus on the related text information for retrieval instead of allowing users intuitively to describe the desired retrieval result. A natural query would for example ask for the top 10 images from the database that are most similar to a fixed image in terms of color, texture, etc.; a query type which is often referred to as 'query by visual example'.

Query optimization needs to be adapted to this essentially different query model for multimedia data. Some systems have already been implemented, e.g. visual retrieval systems like IBM's QBIC [FBF$^+$94] or Virage's VIR [BFG$^+$96]. Database applications and middlewares like GARLIC [CHS$^+$95], VisualHarness [SSPM99], HERMES [SAB$^+$99] or the HERON project [KEUB$^+$98] have already started to use the capabilities of visual retrieval. A major challenge in all of these systems is that similarity between different objects cannot be defined precisely. To handle queries on similarity different kinds of information on the multimedia objects have to be stored. For example in the case of images this could be color histograms, features on textures and layout or related fulltext information describing the object. As similarity cannot be measured exactly, the form of the retrieval result likewise has to be adapted to the user's needs. Consider the following query returning only the top four objects from the HERON-database [KEUB$^+$98], ranked according to their aggregated similarity score (cf. figure 1):

```
SELECT  top(4, images)
FROM    repository
RANK BY average_color(images)
      SIMILAR TO average_color(ex1.pic)
    AND texture(images)
      SIMILAR TO texture(ex2.pic)
```

Queries on similarity do not have to focus on one single feature. In general multimedia queries will refer to at least some different features simultaneously. According to a potentially weighted combining function for each database object an *aggregated score value* is computed. The results are then sorted according to their scores and are returned with a *rank number* – the top-ranked object has the best score value of the entire database collection and so on. A

Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.



Rank	Score	Oid
1	0.91	o4
2	0.88	o5
3	0.87	o1
4	0.69	o7

Figure 1: Query on color and texture with top 4 results

query focusing on only one feature is called *atomic*. Any complex multimedia query can be seen as a combination of atomic subqueries.

One optimization challenge is to combine the ranked results of atomic queries in order to determine the k overall best objects from the database. A naive approach would calculate the aggregated score for all database objects according to a given combining function. With growing size of the database, this obviously results in unacceptable response times, requiring a linear scan of the entire database. But as only the k top objects have to be returned, not all database objects have to be accessed. In this paper we will focus on efficient query combinations of atomic subqueries. We aim on solutions that guarantee a correct result set and at the same time minimize the number of objects to be accessed. Previous significant work in this area is due to Fagin [Fag96, Fag98], who gives an algorithm that guarantees a correct result set. This algorithm is asymptotically optimal in terms of database size with arbitrarily high probability, however only for uniform score distributions – which very rarely occur in practice.

The next section introduces the basic version of our new algorithm, called Quick-Combine. Section 3 extends Quick-Combine to cope with skewed data, which is ubiquitous in practice. In section 4 we will prove analytical results for the worst-case behavior of Quick-Combine. Section 5 reports the speed-up gain of Quick-Combine for skewed data, implying that a real performance breakthrough is achievable. The last section will give a summary of our results and an outlook on parallel work for query combination in heterogeneous environments.

2 A New Test of Termination

In [Fag96] an approach has been presented to process a complex query consisting of several atomic subqueries that may use any monotonous combining function, as for example the maximum or arithmetical mean. This algorithm correctly retrieves the k best objects in the database for any such combination of atomic queries. We will use Fagin's algorithm as a yardstick throughout this paper.

In general atomic queries can be posed in two ways:

- The first type is searching the database and retrieving the objects in the database ordered by descending score for a single feature, which we refer to as enlarging an *atomic output stream* or *sorted access*.

- On the other hand a specific object's score in each atomic output stream could be of interest. This case is referred to as *random access*.

2.1 The Quick-Combine Algorithm (Basic Version)

Fagin's algorithm proceeds in two phases: The first producing atomic output streams and the second consisting of random accesses. Though both phases are necessary, the number of necessary accesses can be minimized with a new test of termination. For this new test we do not only use the information of ranks in output streams, but also the scores which are assigned to all objects in each output stream and the specific form of the combining function.

The following algorithm returns the top answer $(k = 1)$ to any combined query consisting of n atomic subqueries $q_1, ..., q_n$ aggregated using a monotone combining function F. Let x be an object and $s_i(x)$ be the score of x under subquery q_i. An object occuring in the result set of subquery q_i on rank j will be denoted $r_i(j)$.

Algorithm Quick-Combine (basic version):

1. For each subquery compute an atomic output stream consisting of pairs $(x, s_i(x))$ in descending order based on score and get some first elements.

2. For each object output by a stream that has not already been seen, get the missing scores for every subquery by random access and compute its aggregated score $S(x) = F(s_1(x), \dots, s_n(x))$.

3. Check if the present top-scored object o_{top} is the best object of the database:

   Compare the aggregated score $S(o_{top})$ to the value of F for the minimum scores for each subquery that have been returned so far. Test:

   $$S(o_{top}) \geq F(s_1(r_1(z_1)), \dots, s_n(r_n(z_n))) \qquad (1)$$

   where z_i is the lowest rank that has already been seen in the output stream of q_i.

4. If inequality 1 holds, according to theorem 1 o_{top} can be returned as top object of the whole database. If inequality 1 does not hold, more elements of the output streams have to be evaluated. Therefore get the next elements of the streams and proceed as in step 2 with the newly seen objects. □

2.2 Examples and Correctness

Consider the sample results of our query by visual example (cf. fig. 1). The following example will show how to get the top-scored object of the database ($k = 1$) using Quick-Combine:

s_1 : query on texture				
rank	1	2	3	...
score	0.96	0.88	0.85	...
object	o1	o2	o3	...

s_2 : query on avg. color				
rank	1	2	3	...
score	0.98	0.93	0.79	...
object	o4	o5	o6	...

The atomic output streams s_1 and s_2 are evaluated alternately. As objects in both streams are collected one after another, their aggregated score has to be calculated using for instance the arithmetical mean as combining function $F(s_1(o), s_2(o)) = \frac{s_1(o)+s_2(o)}{2}$. Therefore random accesses have to be made:

random accesses	object	o1	o4	o2	o5	...
	output stream	s_2	s_1	s_2	s_1	...
	score	0.78	0.84	0.40	0.83	...

Now the test of termination can be performed using the lowest scores seen in each output stream. Due to the sorting of the streams these scores are the scores of the object that has been seen last in each stream:

test of termination				
last object seen	o1	o4	o2	o5
agg. score	0.87	0.91	0.64	0.88
$F(lowest\ scores)$	-	0.965	0.94	0.905
o_{top}	o1	o4	o4	o4

After accessing the fourth object the evaluation of streams can already be stopped as inequality (1) holds: $0.91 = S(o_{top}) \geq F(s_1(o2)), s_2(o5)) = 0.905$. Now o4 - the best object seen - can be returned as top-scored object of the entire database. Note that none of the objects accessed has been seen by sorted access in both streams.

Quick-Combine is applicable for every monotonous combining function, even in the case of maximum and minimum. In this case Fagin presents two special algorithms differing from his general approach. To show that Quick-Combine's test of termination will definitely return the most relevant object from the database the following theorem is stated:

Theorem 1 (Correctness of results)
If the output streams are evaluated until inequality 1 holds, the object providing the best aggregated score for all objects in any output stream so far has the best aggregated score of all objects in the database. (Proof omitted)

3 Efficiency Improvements for Skewed Data

Now we focus on a further gain of efficiency. We show that evaluating the test of termination twice can save random accesses. We also create a control flow that takes advantage of the distribution of scores in each stream, and address weighted queries. Then we generalize the test of termination to return the k best objects from the database and present the complete algorithm. After proving Quick-Combine's correctness at the end of this section, we will see that the top-scored objects can be successively returned, while the algorithm is still running.

3.1 Reducing the Number of Random Accesses

Without loss of generality we also focus on the case that only the best object of the database will be returned ($k = 1$). Taking a closer look at formula 1, it is obvious that the inequality may become true:

1. If its *right side is reduced*, i.e. any query stream q_j is enlarged and $s_j(r_j(z_j))$ is replaced by $s_j(r_j(z_j+1))$.
2. If its *left side is increased*, i.e. a newly seen object has a maximum aggregated score, sufficient to terminate the algorithm.

In Quick-Combine stream q_j is enlarged first, providing the new score $s_j(r_j(z_j + 1))$. If a new object has been seen, random accesses on $(n-1)$ score values are needed to calculate its aggregated score. The next theorem will show that, if according to case 1 formula 1 already holds before the random accesses are made, the aggregated score of the newly seen object can never be larger than the maximum score of the objects which have already been seen before enlarging q_j. Thus $(n - 1)$ random accesses can be saved, if the test of termination is performed not only after the random accesses, but also before.

Theorem 2 (Saving random accesses)
Let L be the set of objects that have already been seen and whose aggregated scores have been calculated. The aggregated score of any object o_{new} occurring in stream q_j at rank $z_j + 1$ that has been seen last by enlarging query stream q_j, will be less or equal to the maximum aggregated score of objects in L, if formula 1 holds by substituting $s_j(r_j(z_j))$ with $s_j(r_j(z_j + 1))$. (Proof omitted)

3.2 Control Flow for Evaluation of Streams

Quick-Combine considers the top-ranked element of each output stream before proceeding to the next ranked elements. As Quick-Combine uses not only ranks but also scores, a control flow based on the distribution of scores relative to the ranks on which they occur will decrease the number of objects accessed until termination. Of course this distribution has not to be the same in every atomic output stream. We present a heuristic approach to determine in which order streams with different distributions should be evaluated to gain a maximum effect. As a heuristic measure of efficiency a simple rule can be stated:

Accessing less objects to make formula 1 hold means less database accesses and thus results in a more efficient algorithm.

Obviously, there are two ways to make formula 1 hold:

- Initializing the algorithm with some first ranks of each stream helps to *increase the left side*. An object that has been seen later in any output stream generally has to do better in at least one other stream to get the maximum aggregated score, i.e. the chance that it has already been seen on the first few ranks in a different output stream is getting more and more probable. Thus, before a certain query stream should be preferred for further evaluation, it is advisable to analyze some first objects of each stream.

- To *decrease the right side* quickly consider the distribution of scores relative to the ranks on which they occur. This distribution can totally differ in each output stream. Though in all output streams the scores are falling monotonously with declining ranks, there may be streams where the scores only slightly change with decreasing ranks. Streams starting with high score values but declining rapidly may exist or even output streams with scores not changing at all. As we want to force the decline of the right side, streams showing a behavior of declining scores most rapidly relative to the ranks should be preferred for evaluation.

For a more efficient combining algorithm a control mechanism preferring the evaluation of rapidly declining output streams is needed. An obvious measure is the derivative of functions correlating score values to the ranks on which they occur for each output stream. Since these functions are discrete, their behavior can be estimated using the difference between the p^{th} last and the last output score value assuming that there are at least p elements in the stream. Of course the same p has to be used for any stream to provide comparability. A larger value for p better estimates an output stream's global behavior, small values detect more local changes in a stream.

The above considerations are not only useful for equally weighted queries. An indicator for streams with low weights should naturally be regarded less important than indicators for highly weighted streams which should get prior evaluation. As the weights can be expressed in the combining function F (e.g. a weighted arithmetical mean), a simple measure for the importance of each stream q_i is the partial derivative of the combining function $\frac{\partial F}{\partial x_i}$. Thus in the weighted case an indicator for any stream q_i containing more than p elements can be calculated as follows:

$$\Delta_i = \left| \frac{\partial F}{\partial x_i} \right| \cdot (s_i(r_i(z_i - p)) - s_i(r_i(z_i))) \qquad (2)$$

3.3 The Quick-Combine Algorithm (Full Version)

Now we generalize Quick-Combine to a result set containing the k best matches for any $k \in \mathbb{N}$ and implement our control flow. Under the assumption that there are at least k objects in each stream, it returns the top k answers to any complex query consisting of n atomic subqueries $q_1, ..., q_n$. For each subquery a ranked result set consisting of pairs $(x, s_i(x))$ in descending sorted order based on score is computed where x is an object and $s_i(x)$ is the score of x under subquery q_i.

Algorithm Quick-Combine (full version):

0. *Initialization:* Get the first p results for each subquery, where p is a suitable natural number. Compute an indicator Δ_i for each query stream q_i according to equation 2.

1. *Random access for new objects:* For each new object output by any stream that has not already been seen previously, get the missing scores for every subquery by random access. For each new object there are $(n-1)$ random accesses necessary. Objects that have already been seen before can be ignored.

2. *Calculation of aggregated scores:* For any new object x that has been seen compute the aggregated score $S(x) = F(s_1(x), \ldots, s_n(x))$.

3. *First test of termination:* Check if the k top-scored objects are already in what has been seen so far: Compare the aggregated score of the present k top-scored objects to the value of the combining function with the lowest scores seen for each feature. Check if there are at least k objects whose aggregated score is larger or equal than the aggregated minimum scores per feature:

   $$|\{x | S(x) \geq F(s_1(r_1(z_1)), .., s_n(r_n(z_n)))\}| \geq k \qquad (3)$$

   If inequality 3 holds, according to theorem 3 the k top-scored objects can be returned as top objects of the whole database.

4. *Enlarging an atomic output stream:* If inequality 3 does not hold, more elements of the atomic output streams have to be evaluated. Therefore get the next element of the stream having the maximum Δ (if the maximum Δ is reached for two or more streams any of them can be chosen randomly).

5. *Second test of termination:* Check if inequality 3 holds using the new object's score. If the inequality holds, return the k top-scored objects.

6. *Indicator computation:* Calculate a new Δ for the enlarged stream. Proceed as in step 1 with the newly seen object. □

As a control mechanism the indicator Δ approximates the local behaviour of the distribution of absolute score values relative to the ranks on which they appear. Again we will have to show that no relevant database object is missed by Quick-Combine:

Theorem 3 (Correctness of Results)
If the output streams are evaluated until inequality 3 holds, the k objects providing the best aggregated scores that appeared in any output stream so far have the best aggregated score of all objects in the database.

Proof: *It is to show that no object that has not been seen can have a larger aggregated score than the top k objects that have been seen.*

Let inequality 3 hold, x be any of the top k objects and o be an object from the database that has not been seen yet in any of the output streams. Then due to the sorting of the streams the atomic scores of o satisfy

$$s_1(o) \leq s_1(r_1(z_1)), \ldots, s_n(o) \leq s_n(r_n(z_n))$$

and therefore due to the monotony of F and formula 3:

$$\begin{aligned} S(o) &= F(s_1(o), \ldots, s_n(o)) \\ &\leq F(s_1(r_1(z_1)), \ldots, s_n(r_n(z_n))) \\ &\leq F(s_1(x), \ldots, s_n(x)) = S(x). \end{aligned} \quad \square$$

Theorem 3 shows that the first found object satisfying inequality 3 always is the top-scored object of the whole collection. Thus it can be delivered to the user as soon as it is found, which of course also applies to all following ranks up to k, when the algorithm finally terminates. If the user is already satisfied by the first few ranks, the query execution for large values of k can be stopped during processing. We therefore state the following corollary to theorem 3:

Corollary 1 (Successive Output of Results)
Since Quick-Combine will run until k objects that satisfy formula 3 are successively found, for $k > 1$ the first objects can already be returned while the algorithm is still running.

4 Complexity for Uniform Distributions

In this section we focus on the efficiency of Quick-Combine compared to Fagin's algorithm. In particular we will show that Quick-Combine's complexity is upper-bounded by the complexity of Fagin. We give a general geometrical interpretation of efficiency issues and present improvement factors even in the rare case of uniform distribution of score values.

4.1 Worst Case Complexity

The following theorem will show that Quick-Combine will never access more distinct objects than Fagin's algorithm. To this end the number of distinct objects that Fagin's algorithm collects in its first phase is compared to the number of distinct objects collected by Quick-Combine.

Theorem 4 (Upper-Bounding)
Given n atomic output streams sorted in descending order. Formula 3 holds, if all streams have been evaluated at least as far enough that Fagin's algorithm terminates, i.e. that there is a set L of k objects delivered by all streams.

Proof: *Let $o_1, \ldots, o_k \in L$ be k different objects that have been output by each stream and F be any monotonous combining function. Due to the descending order of scores in every stream any atomic score $s_i(o_j)$ $(1 \leq i \leq n, 1 \leq j \leq k)$ satisfies:*

$$s_1(o_j) \geq s_1(r_1(z_1)), \ldots, s_n(o_j) \geq s_n(r_n(z_n))$$

and thus due to the monotonocity of F:

$$\begin{aligned} S(o_j) = F(s_1(o_j), \ldots, s_n(o_j)) \geq \\ F(s_1(r_1(z_1)), \ldots, s_n(r_n(z_n))) \end{aligned}$$

for each of the objects $o_1, \ldots, o_k$, i.e. equation 3 holds. $\square$

4.2 Geometrical Interpretation

According to [PF95], a geometrical model for combining query results could be as shown in figure 2 for the case $n = 2$. If each atomic subquery is mapped onto an axis divided into score values from 0 (no match) to 1 (exact match), each object in the database can be represented by a point in n-dimensional space.

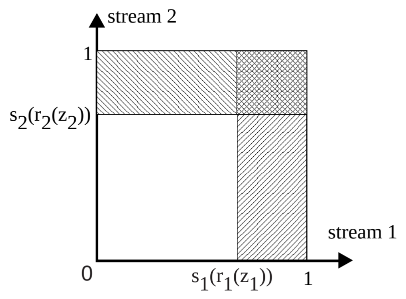


Figure 2: Combining two atomic output streams

Evaluating results of an atomic subquery by ranks can be represented by moving a hyperplane orthogonal to its axis from 1 downwards to 0. The order in which objects are collected by the hyperplane can exactly be mapped onto the ranks on which they occur. For example consider figure 2 and assume that output stream i for subquery i has been evaluated for all objects o that satisfy $s_i(o) \geq s_i(r_i(z_i))$, i.e. all objects have been retrieved from stream i, whoses scores is larger or equal to the score of the object occurring on rank z_i in stream i $(i = 1, 2)$. The areas evaluated from stream 1 and stream 2 have an intersection containing the top-scored objects already seen in both streams. Fagin's algorithm in its first phase collects k objects that occur in the dark-shaded area. Of course all the objects collected in the shaded areas need random accesses, as they all have been seen in the first phase.

Evaluations of aggregated scores by a *monotonous* combining function can also be represented by moving a hypersurface collecting objects while moving over the area. As the k objects collected first should have the top aggregated scores and thus can be returned as correct retrieval result, the hypersurface has to be orthogonal to the optimal direction starting at the optimal level.

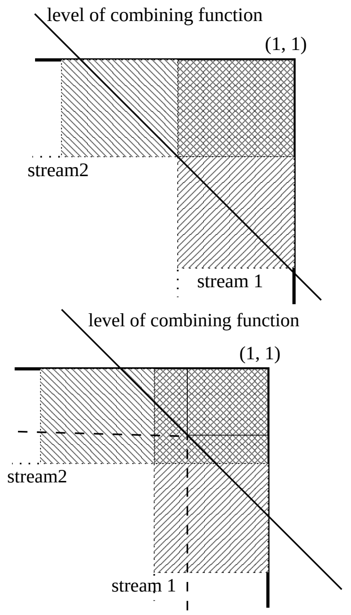


Figure 3: The arithmetical mean as combining function for Fagin's algorithm (upper) and Quick-Combine (lower)

To return a correct result the hypersurface has to collect the first k objects. Since Fagin's algorithm insists that there are at least k objects in the dark-shaded area, the hypersurface only sweeps objects with already calculated aggregated scores. Thus no relevant object can be missed. But depending on the combining function the area and thus the number of objects in Fagin's algorithm, for which aggregated scores are calculated, might be far too large. Consider for example figure 3 (left) showing the arithmetical mean as combining function. The hypersurface can collect any object till the lower left corner of the dark-shaded area, as all objects in this area have been seen and also all aggregated scores for these objects have been calculated.

There are at least k objects in the dark-shaded area, but also all the objects in the two light-shaded triangles between the hyperplane and the dark-shaded area are collected. For e.g. uniform distributions a triangle with the same area as the dark-shaded area would also guarantee the correctness of results, but would minimize the calculations of aggregated scores and the random accesses needed, cf. figure 3 (right). Unlike Fagin's algorithm Quick-Combine only concentrates on this minimized area. As shown in figure 3 (right) for the case $n = 2$, in Quick-Combine streams 1 and 2 would only be enlarged down to the dashed lines.

4.3 Improvement Analysis for Uniform Distributions

Fagin has proven the important result that his algorithm is expected to be asymptotically optimal for independent output streams under uniform score distribution. Though uniform distributions rarely occur in practice, it can be stated that also in this case Quick-Combine improves Fagin's algorithm by minimizing the number of object accesses.

For the analysis our geometrical interpretation is used. With growing dimension, i.e. the number of atomic subqueries to be combined, the dark-shaded area in our model evolves to a high dimensional cuboid whose volume determines the number of objects to calculate aggregated scores for. Depending on the combining function also in high dimensional cases this cuboid is contained by a geometrical figure guaranteeing correctness for more than k objects.

Consider the n-dimensional case with the arithmetical mean as combining function and the output streams evaluated down to score s. Then the triangles of figure 3 have to be generalized to polyhedra $S_{n,s}$, the dark-shaded square to a n-dimensional cube $W_{n,s}$ and the light-shaded rectangles to n-dimensional cuboids. The polyhedron $S_{n,s}$ is formed by the set of all points on or above the combining function's hypersurface $\frac{1}{n}\sum_{i=1}^{n} x_i = s$. The next theorem shows that the cube's volume shrinks rapidly with growing dimensions in proportion to the polyhedron's volume.

Theorem 5 (Ratio between Cube and Polyhedron) *Let W_n be the n-dimensional cube in $[0,1]^n$ with $W_n = \{(x_1,\ldots,x_n) \,|\, 0 \le x_i \le \frac{1}{n}$ for $i = 1,\ldots,n\}$ and $Vol(W_n)$ be its volume. Let further S_n denote the polyhedron $S_n = \{(x_1,\ldots,x_n) \,|\, 0 \le \frac{1}{n}\sum_{i=1}^{n} x_i \le \frac{1}{n}\}$ and $Vol(S_n)$ be its volume.*

Then the ratio $\frac{Vol(W_n)}{Vol(S_n)}$ is equal to $\frac{n!}{n^n}$. (Proof omitted)

To get a more precise impression of the efficiency gain one has to compare the total number of objects accessed by Fagin's algorithm and Quick-Combine. Therefore the volume of the dark- and light-shaded areas (cf. figure 2) of Fagin's algorithm has to be compared to the corresponding areas needed by Quick-Combine using a polyhedron of the same volume as Fagin's cube. Increasing volumes of the cuboids obviously causes more necessary object accesses and the ratio between these volumes is also the improvement factor for the number of objects accessed.

Theorem 6 (Improvement for Uniform Distributions) *The total number of objects accessed by Quick-Combine using the polyhedron $\tilde{S}_n$, which has the same volume as the cube W_n needed by Fagin's algorithm, is $(\frac{n}{\sqrt[n]{n!}})$ times smaller than the number of objects accessed by Fagin's algorithm for uniformly distributed scores. (Proof omitted)*

With Stirling's formula the improvement factor is asymptotically equal to $\frac{n}{\sqrt[n]{n!}} \approx \frac{e}{\sqrt[n]{\sqrt{2\pi n}}} \overset{(n\to\infty)}{\longrightarrow} e$. Quick-Combine thus results in an efficiency gain of 2.72 with increasing n. As not only visual features determine a multimedia query, but also ranked results from text retrieval, typical values for n are between five and ten. Table 1 shows improvement factors for some practical values of n:

n	3	4	5	6	7	8	9
	1.64	1.81	1.92	2.01	2.07	2.13	2.17

Table 1: Improvement factors for object accesses in the uniformly distributed case.

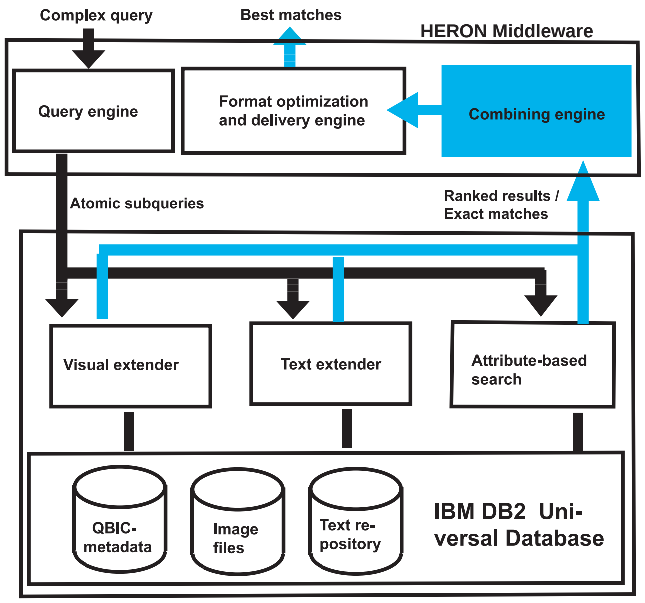


Figure 4: Architecture of the HERON system

5 Speed-up Results for Skewed Data

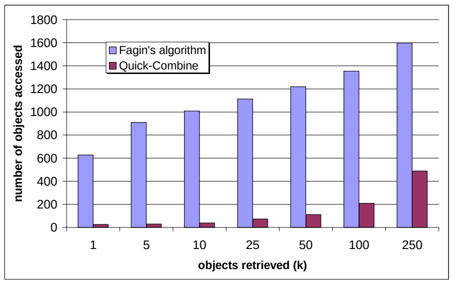
Fagin's optimality results – plus the improvements of Quick-Combine – are only valid for the very unlikely case of uniform score distributions. In practice, however, skewed data is prevalent. Thus the name of the game is about efficiency speed-ups for such practical, skewed data. We will first report performance results from practical data, followed by extensive synthetic benchmarks.

5.1 Benchmark Results for Practical Data

The HERON system [KEUB+98] has been used to process a set of atomic queries on heraldic images. Besides components for query composition, user specific image delivery and efficient format conversions, it features a combining engine that can use different visual retrieval systems and databases (cf. fig. 4). The combining engine implements Quick-Combine. For our experiments we used IBM DB2 V 5.2 and QBIC technology [FBF+94] for visual retrieval.

To measure the gain of efficiency the number of objects to be retrieved was compared to the average number of objects which the combining algorithm had to access. As described in section 4 the number of objects accessed determines the number of random accesses that are needed to calculate aggregated scores and thus forms the main computational costs. To be exact, Fagin's algorithm will not need random accesses for all the objects, but as k objects have already been seen in all atomic output streams, their aggregated values can directly be calculated. Nevertheless, those small number of random accesses have proven to be rather negligible even in the case $n = 3$.

We set up a scenario for the combination of three atomic subqueries over a repository of 2300 heraldic images from the HERON database. The randomly chosen subqueries focused on the image's average color, textures and color histograms, i.e. $n = 3$; as combining function we chose the arithmetical mean and used an indicator computation for $p = 3$. Figure 5 shows the average experimental results for 30 different queries. The output streams were statistically independent as e.g. the color of an object is not supposed to be related to its shape or texture.The number of objects accessed is plotted against the number of objects k to be returned. Since the scores are *not distributed uniformly*, Fagin's algorithm accesses far more objects. Obviously, the early use of the combination function's composition and the use of our control flow in Quick-Combine results in a higher gain of efficiency especially for larger values of k. For practical values of k ($k \leq 50$) Quick-Combine even accesses *30 times less objects* than Fagin's algorithm.



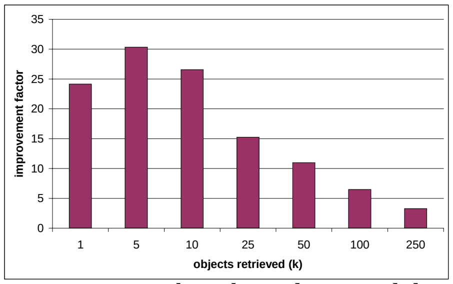
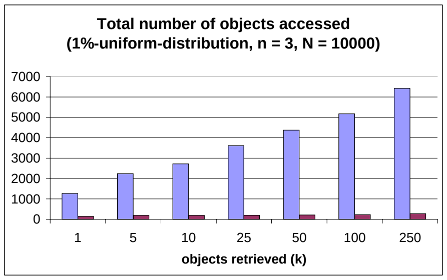


Figure 5: Benchmark results on real data

5.2 Benchmark Results for Synthetic Data

From our practical tests we gained insight into distributions that really occur in image retrieval. We argue that typical distributions from visual retrieval systems are a low percentage of objects having high and medium score values and a high percentage having very low scores. If text retrieval is included the percentage having high and medium scores even decreases. We extensively tested these types of distributions on synthetic data for two databases with $N = 10000$ and $N = 100000$ objects generating different score distributions. The performance of Quick-Combine changes with variations of the number k of objects to return, the number n of streams to combine, the database size N and the skewedness of the distribution. The results of Quick-Combine are always compared to the result of Fagin's algorithm. The efficiency measures to compare are threefold: The number of distinct database objects accessed, the number of necessary sorted accesses and the number of necessary random accesses.



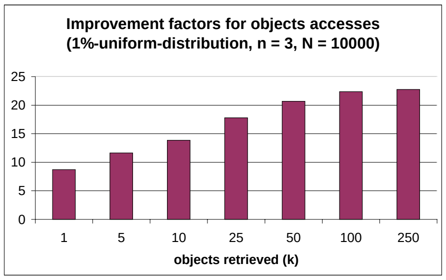
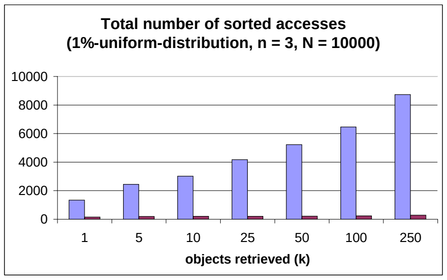


Figure 6: Average number of object accesses for skewed distributions

Our first test scenario focused on a slightly skewed score distribution typical for content-based image retrieval. One percent of database objects have high or medium score values (uniformly distributed), the rest shows values smaller than 0.1. On the smaller database ($N = 10000$) we combined three streams ($n = 3$). As can be seen in the diagrams (Fig.6, Fig.7 and Fig.8) we measured the number of accesses of different objects and the number of sorted and random accesses for varying values of k. Fagin's algorithm (light columns) on the average always needs to access far more objects than Quick-Combine (dark columns). On the right diagram the respective average improvement factors can be seen. For all three types of accesses they range between 10 and 20. Obviously the improvement scales with varying k as for $k = 250$ already 2.5% of the entire database size is retrieved.

Observation: In *all our experiments* the ratio of sorted and random accesses between Fagin's algorithm and Quick-Combine was nearly the same as the respective ratio of distinct object accesses. Thus in the further analysis we will concentrate on these accesses and omit the diagrams for sorted and random accesses.

The next experiments focused on even more skewed score distributions (Fig. 9). A score distribution of 0.1% high and medium scores and 99.9% of low scores was generated. Here average improvement factors around 100 can be observed for $k \leq 25$ as Quick-Combine adopts itself to the specific distribution. The improvement for $k \geq 50$ in



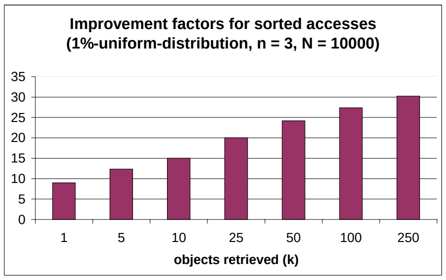


Figure 7: Average number of sorted accesses for skewed distributions

this case is minimal since with $N = 10000$ and $n = 3$ there are only 30 objects in the database having noticable scores.

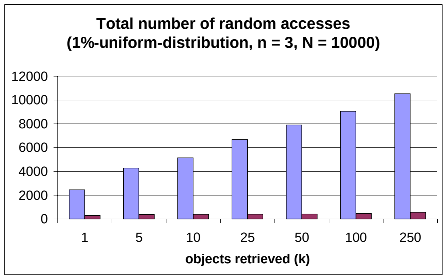
The next diagram (Fig.10) shows the scalability to large databases. A database with $N = 100000$ was generated showing the same score distribution as above. Note that for the retrieval of 0.25% of database size Quick-Combine accesses little objects, whereas Fagin's algorithm already accesses a third of the entire database objects. Average improvement factors in this case range from 50 to 120.

The last experiment (Fig. 11) analyzes the scalabilty of Quick-Combine, if a varying number n of streams is combined. We combined up to 10 different streams using the same database size and score distribution as in our first experiment. Here we observed average improvement factors ranging from 10 to 20. Note that Fagin's algorithm accesses almost all database objects if more than 5 output streams are combined.

5.3 Discussion of Overall Performance Results

As stated in [Fag96] Fagin's algorithm is expected to access $k^{\frac{1}{n}} N^{(1-\frac{1}{n})}$ objects. As shown before Quick-Combine is expected to access $(\frac{\sqrt[n]{n!}}{n})$ less objects in the uniformly distributed case. But this case is very rare in practice. To get an impression on practical efficiency issues the experiments on real or synthetic data with more practical score distributions had to be compared.

- In all our experiments with real and synthetic data the number of objects Fagin's algorithm accesses is by far higher than the number accessed by Quick-Combine.
- The number of sorted and random accesses in Fagin's



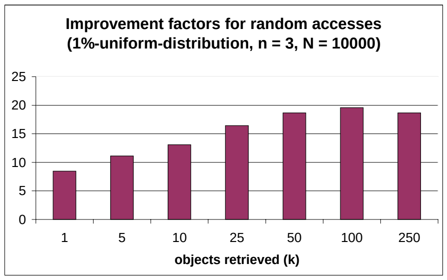
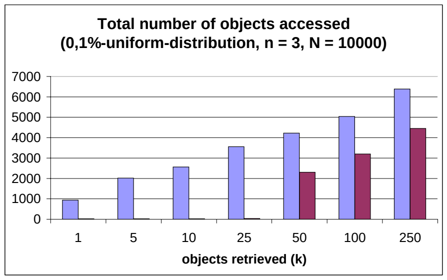


Figure 8: Average number of random accesses for skewed distributions



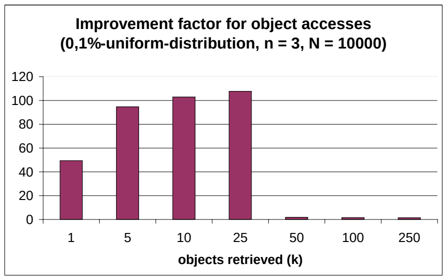


Figure 9: Average number of object accesses for very skewed distributions

algorithm is also always considerably higher than in Quick-Combine.

- Quick-Combine scales both with growing values for k and with increasing number n of streams to combine.
- Quick-Combine is very efficient even for large database sizes.
- Quick-Combine is also highly efficient for very skewed score distributions.

Fagin's work also strongly influenced statistical approaches as [CG97], for which experimental results show that the number of objects retrieved can be considerably smaller than in Fagin's algorithm. However, such approaches gain performance in exchange of a guaranteed correctness of results. With Quick-Combine we can get both: High performance and correct results.

6 Summary and Outlook

In this paper we proposed an algorithm – called Quick-Combine – to combine multi-feature queries that are typical for the use in modern digital libraries and digital image archives. We examined previous work in this area and compared our approach to Fagin's algorithm presented in [Fag96]. Quick-Combine efficiently retrieves the k most relevant database objects with guaranteed correctness for every monotonous combining function. Measuring the number of necessary database accesses, our theoretical analysis as well as experimental results indicate that Quick-Combine is considerably more efficient than Fagin's algorithm. This speed-up of Quick-Combine compared to Fagin's algorithm increases with growing skewedness of score distributions from a factor around 2 towards one or two orders of magnitude. A real live benchmark with heraldic images showed speed-up factors around 30.

These remarkable benchmark results suggest that with Quick-Combine a real performance breakthrough for content-based image retrieval in large image databases is achievable. Definitely so, high-speed iterators for sorted accesses (relying on efficient multi-dimensional indexes) and fast access methods for random accesses are essential. Parallel execution of several sorted streams (e.g. by Java threads) can be done in addition. Given all of that fast Internet access to very large image databases, like commercial digital photo archives, is in sight now.

Acknowledgements

We are grateful to Matthias Wagner, Thomas Birke and Achim Leubner for helpful comments and suggestions. The HERON-project is funded by the German Research Foundation DFG within the strategic research initiative ”Distributed Processing and Exchange of Digital Documents”. The European patent is supported by the DFG Ideenwerkstatt. The database software applied in the HERON system is granted within IBM's program ”DB2 For Educational Purpose”.

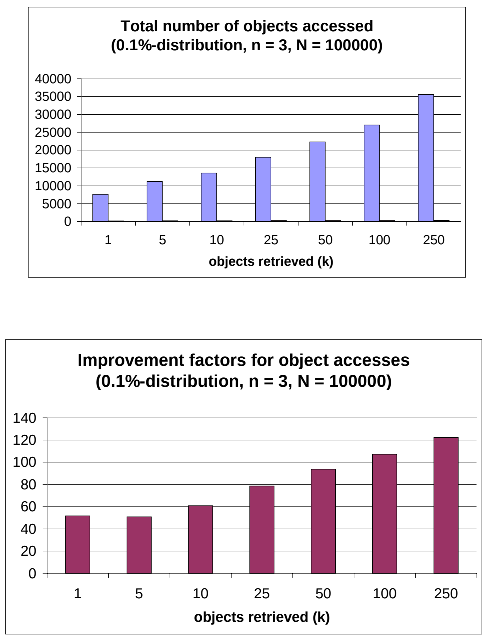


Figure 10: Average number of object accesses for large databases

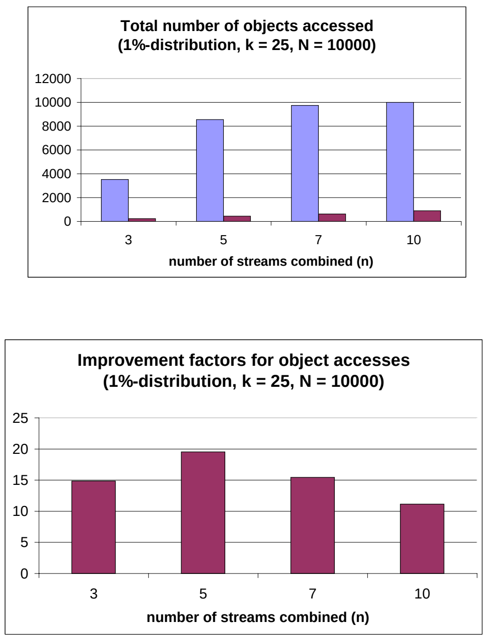


Figure 11: Average number of object accesses with varying number of streams to combine

References

[BFG+96] J. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Humphrey, R. Jain, and C-F. Shu. Virage image search engine: An open framework for image management. In *Storage and Retrieval for Image and Video Databases (SPIE) 1996*, pages 76–87, 1996.

[CG97] S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. In *16th ACM Symposium on Principles of Database Systems*, pages 91–102, Tucson, 1997. ACM.

[CHS+95] M. Carey, L. Haas, P. Schwarz, M. Arya, W. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. Williams, and E. Wimmers. Towards heterogeneous multimedia information systems: The GARLIC approach. In *5th Intern. Ws. on Research Issues in Data Engineering: Distr. Object Management*, pages 124–131. IEEE-CS, 1995.

[Fag96] R. Fagin. Combining fuzzy information from multiple systems. In *15th ACM Symposium on Principles of Database Systems*, pages 216–226, Montreal, 1996. ACM.

[Fag98] R. Fagin. Fuzzy queries in multimedia database systems. In *17th ACM Symposium on Principles of Database Systems*, pages 1–10, Seattle, 1998. ACM.

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[KEUB+98] W. Kießling, K. Erber-Urch, W.-T. Balke, T. Birke, and M. Wagner. The HERON project — multimedia database support for history and human sciences. In J. Dassow and R. Kruse, editors, *Proc. of the GI Annual Conf. INFORMATIK'98*, pages 309–318. Springer, 1998.

[PF95] U. Pfeifer and N. Fuhr. Efficient processing of vague queries using a data stream approach. In *18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 189–197, Seattle, 1995. ACM.

[SAB+99] VS. Subrahmanian, S. Adali, A. Brink, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. HERMES: Heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1999.

[SSPM99] A. Sheth, K. Shah, K. Parasuraman, and S. Mudumbai. Searching distributed and heterogeneous digital media: The VisualHarness approach. In *8th Conf. on Database Semantics - Semantic Issues in Multimedia Systems*, pages 311–330. Kluwer, 1999.