Efficient Combination of Ranked Result Sets in Multi-Feature Applications

Wolf-Tilo Balke

University of Augsburg, Germany

balke@informatik.uni-augsburg.de

© Copyright 2001. All rights reserved.
Abstract

Applications like multimedia databases or enterprise-wide information management systems have to meet the challenge of efficiently retrieving best matching objects from vast collections of data. For instance, in 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. In the recent years combining algorithms have been proposed to cope with this essentially different retrieval model.

Generally speaking, we distinguish three environments for the combination of ranked results. In homogeneous environments the various features are used on a set of objects that can be identified by a common key. The quasi-homogeneous environment uses features on different collections of data that share some common, standardized attributes. The last and rather rare case are heterogeneous environments, where objects from different collections have to be compared using a complex function.

We present a new combining algorithm called Quick-Combine for combining multi-feature result lists in (quasi-) homogeneous environments, 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 common algorithms 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.
Also for heterogeneous environments we present an innovative algorithm called Stream-Combine for processing multi-feature queries on heterogeneous data sources. This algorithm can guarantee the correct retrieval of the $k$ top-ranked results without using any random accesses. Stream-Combine implements sophisticated heuristics and therefore is self-adapting to different data distributions and to the specific kind of the combining function. Furthermore we present a new retrieval strategy that will essentially speed up the output of relevant objects.

As benchmarks on practical data promise that our combining algorithms – both protected by European patent No. EP 00102651.7 (patent pending) – can dramatically improve performance, we also want to discuss interesting applications of the combination of ranked result sets in different areas. The applications for the optimization in ranked query models are manifold. Generally speaking we believe that all kinds of federated searches in database or portal technology can be supported like e.g. content-based retrieval, knowledge management systems or multi-classifier combination.
Contents

1 Introduction 11
  1.1 Motivation of the Combining Problem: Multimedia Databases . . . 13
  1.2 Definition of the Combining Problem ............................ 15

2 Combination of Ranked Result Sets in Homogeneous Environments 17
  2.1 The Framework for Multi-Feature Optimization .................. 17
    2.1.1 Fagin’s algorithm ........................................ 17
    2.1.2 Statistical approaches .................................... 18
    2.1.3 A New Test of Termination and a Basic Version of the Quick-
          Combine algorithm ........................................ 20
  2.2 Further Improvements of Quick-Combine .......................... 22
    2.2.1 Optimization of Random Accesses .......................... 22
    2.2.2 Control Flow for Evaluation of Streams .................... 24
    2.2.3 Generalization for Result Sets ............................. 25
  2.3 Complexity Analysis of Quick-Combine .......................... 28
    2.3.1 Worst Case Analysis ...................................... 28
    2.3.2 Geometrical Interpretation of Efficiency Issues .......... 28
    2.3.3 Improvement Analysis for Uniform Distributions .......... 31
  2.4 Experimental Results Using Quick-Combine ..................... 33
    2.4.1 Benchmark Results for Practical Data ....................... 34
    2.4.2 Benchmark Results for Synthetic Data ..................... 35
    2.4.3 Discussion of Results .................................... 38
  2.5 Successive Retrieval of Objects Using Quick-Combine ........... 40

3 Stream-Combine – An Algorithm for Heterogeneous Environments 45
  3.1 Related Work .................................................. 45
3.2 Algorithm Stream-Combine (Basic Version) .................. 47
3.3 Improvements for Stream-Combine .......................... 50
  3.3.1 Retrieving the Set of k Overall Best Objects ........... 50
  3.3.2 Optimizing the Choice of Streams for Further Expansion .. 51
  3.3.3 Pruning the Search Space ................................ 53
  3.3.4 Algorithm Stream-Combine (Full Version) ............. 53
  3.3.5 Using Lower Bounds for Stream-Combine ............... 56
3.4 Efficiency Issues of Algorithm Stream-Combine ............ 58
  3.4.1 Worst Case Estimation .................................. 58

4 Applications in Visual Retrieval Systems .................... 61
  4.1 Content-based Retrieval ..................................... 62
  4.2 The SQL/MM Standard ....................................... 64
    4.2.1 A Datatype for Still Images ............................ 64
    4.2.2 Methods for Manipulating Images in SQL/MM .......... 65
    4.2.3 Searching Images with SQL/MM ....................... 66
  4.3 Commercial Content-based Retrieval Systems and Research Prototypes 68
    4.3.1 Commercial Products .................................. 69
    4.3.2 Research Prototypes .................................... 70
  4.4 Case Study: Application of Content-based Retrieval in the HERON-project .......... 72
    4.4.1 Middleware-Architecture of the HERON-System .......... 73
    4.4.2 Heraldry and Typical Query Profiles in HERON .......... 77
    4.4.3 Content-based Features for Heraldic Applications .... 81
    4.4.4 Adaption of Retrieval Features to Diverse Image Archives . 84

5 Applications in Multi-Classifier Combination ................ 89
  5.1 The Problem ................................................ 90
  5.2 Using Combining Algorithms for Multi-Classifier Systems .... 91
  5.3 Case Study: Speech Recognition ......................... 93
    5.3.1 The Multi-Feature Speech Recognition Problem ....... 94
    5.3.2 Integration of Dictionaries within the Stream-based Approach 95

6 Applications in Information Portals .......................... 97
  6.1 A New Challenge ........................................... 98
  6.2 Federated Searches ......................................... 101
7 Summary and Outlook

7.1 Summary ........................................ 111
7.2 Outlook and Future Work ........................ 113
Contents
List of Figures

1.1 Query on average color and texture with top 4 results .................. 14
2.1 Scheme of Fagin’s algorithm A1 ........................................ 19
2.2 Scheme of algorithm Quick-Combine .................................... 27
2.3 Combining two atomic output streams ................................. 29
2.4 Hyperplane for the arithmetical mean as combining function for Fa- gin’s algorithm A1 (left) and the algorithm Quick-Combine (right) .... 30
2.5 Benchmark results on real data ......................................... 35
2.6 Average number of object accesses for skewed distributions ....... 36
2.7 Average number of sorted accesses for skewed distributions ....... 36
2.8 Average number of random accesses for skewed distributions ...... 37
2.9 Average number of object accesses for very skewed distributions .. 37
2.10 Average number of object accesses for large databases ............ 38
2.11 Average number of object accesses with varying number of streams to combine ......................................................... 38
2.12 Successive output for uniform score distributions ................. 40
2.13 Successive output for skewed score distributions .................... 41
2.14 Successive output for very skewed score distributions ............. 42
2.15 Successive output of results for different score distributions ....... 43
4.1 Architecture of the HERON system .................................... 74
4.2 Typical user interaction with the HERON system ...................... 77
4.3 Image of a shield with blazoning from Siebmacher: 'Siebmachers grosses Wappenbuch' ................................................. 79
4.4 17th Century portrait and magnification of the upper right corner .. 80
List of Figures

4.5 16th Century painting showing the arms of the person portrayed (upper left corner) and the later inserted arms of a collector (upper right corner) ........................................... 81

4.6 Query by texture ........................................... 82

4.7 Query by shape: geometric shape ....................... 83

4.8 Query by shape: complex shape ......................... 84

4.9 Query result on Texture for heraldry (vertically shaded) ........ 86

4.10 Query result on texture for ornaments (stylized leaves) ........ 87

4.11 Query result on texture for ornaments (geometrical) .......... 88

6.1 Deriving a traffic prognosis from quasi-homogeneous sources .... 108
1 Introduction

The development of multimedia databases, digital libraries and knowledge management systems has become a wide area of research in recent years. All of these systems have in common that they need to process a large amount of information, that even may be distributed over a variety of data sources. Processing these data requires tasks like storing data, retrieving it from the data source and delivering it to the user taking specific user profiles into account. However, these tasks cannot be solved without having a closer look at the field of application. Especially if multimedia data is involved, simple tasks for merely text-based applications (like e.g. the retrieval) can grow into most complex operations, that may need a lot of advanced techniques.

To make large multimedia collections accessible to a wide variety of users providing content via the Internet is a necessary step. Besides copyright restrictions, the pure amount of data often cannot be stored on one single server, bandwidths may be too small and answer times too long. Thus data has to be distributed over several servers and can then be shared effectively by a variety of users. Recently a new technology has been developed to stand at the front end of such shared data repositories: The portal. Portals provide users with a personalized gateway to distributed data satisfying their information needs. Portals are a blend of software applications that at the beginning were merely used to access relational databases and return Internet links in web search engines. However, the demands in general information systems quickly escalated the need for more advanced features. Portals in these environments go beyond providing simple connectivity, but have become important tools for updating or adding new content, composing complex queries on the collections and publishing retrieval results in an easy to understand way.

Portal applications together with databases and Internet search engines have become a common ground for a wide variety of purposes. It is needed for such diverse applications like sharing enterprise information and documents within a company for
collaborative working or decision making, enabling joint research and remote diagnoses, e.g. in medical sciences, or opening up archives of cultural heritage for the interested public. For today’s use of information we can state two theses that are fundamental for the work presented in the sequel:

- Getting information for high-level tasks needs *complex queries* that usually involve several parts describing different aspects of the problem. These queries will often rely on similarity of objects or documents using a ranked query model.

- In general useful information will have to be *extracted from and/or assembled of* data gathered from a variety of sources. It may be distributed over several collections and may include all types of data objects, e.g. multimedia data.

However, working with complex queries over different repositories is a exceedingly difficult task. With the emerging need of digitized collections containing compound documents consisting of text, images or video and audio data, also the necessity of retrieval systems that can capture the user’s intention in a most intuitive way has become more and more demanding. The multimedia information explosion together with constantly decreasing costs for both hardware and software components, has rapidly increased the amount of digitally stored and managed data. For instance applications in geographic information systems, medical imaging or art collections provide masses of image data to be stored, retrieved and delivered. Some of those large image repositories like NASA’s Earth Observing System providing a collection of satellite images are collecting a terabyte of data each day [SFGM93]. The efficient usage of such repositories require new sophisticated techniques.

*One of the most important problems is how to merge the (often ranked) result sets that are delivered from different data sources efficiently in order to answer a complex query possibly involving various media types.*

To investigate such modern multimedia information retrieval, we will assume an architecture that has a variety of (possibly heterogeneous) information sources connected with an appropriate user interface or portal application. The connection between information sources and the interface is given by a special multimedia middleware to manage the collection’s contents. In this discourse we will focus on techniques to merge ranked result sets in different environments. We will present combining algorithms to gain considerable speed ups over today’s retrieval systems.
In this section we will take a closer look on multimedia databases as a case study to support our above theses. Multimedia databases are already used for a variety of applications \cite{Fur96, Pra97, Sub98}. We will show that complex queries often are necessary to gain useful results and what is meant by different 'aspects' of a query. Multimedia databases also are often distributed and may involve several collections of media.

In object-relational database systems or information portals the handling of complex documents or multimedia data such as images, video or audio files, demands an entirely new retrieval model different from the model of classical relational database technology. Especially in the field of query evaluation it is typically not a set of objects exactly matching the query that will be retrieved but rather a ranked set of results, where a grade of match is attached to each object returned. This allows the user to specify particular needs and complex queries in a more intuitive way. Another main advantage of information portals or multimedia database systems is returning only a few top-ranked results instead of a probably empty or too large set of objects. This ranked result set is also almost generally used in information retrieval systems.

Imagine for instance 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 e.g. color, texture, etc.; a query type which is often referred to as 'query by visual example'.

Of course systems have to be adapted to this essentially different query model for multimedia data and some systems have already been implemented, e.g. multimedia retrieval systems like IBM’s QBIC \cite{FBF+94} or Virage’s VIR \cite{BFG+96}. Database applications and middlewares like GARLIC \cite{CHS+95}, VisualHarness \cite{SSPM99}, HERMES \cite{SAB+99} or the HERON project \cite{KEUB+98} have already started to use the capabilities of these advanced retrieval capabilities. However, a major problem in all of these systems is that similarity between different objects cannot be defined
1 Introduction

precisely by one single property but will rely on several aspects. In order to handle queries on similarity different information – so-called features – 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 free-text information describing the object.

As similarity cannot be measured exactly, the ranked form of the retrieval result is very suitable to satisfy the user’s needs. Consider the following model query returning only the best four objects from the database, ranked according to their similarity in terms of average color and texture (cf. figure 1.1):

```
SELECT top(4, images)
FROM repository
RANK BY average_color(images)
SIMILAR TO average_color(example1.pic)
AND texture(images)
SIMILAR TO texture(example2.pic)
```

In general queries on similarity do not have to focus on one single feature and almost always multimedia queries will refer to at least some different features simultaneously. According to a potentially weighted combining function for each database

Figure 1.1: Query on average color and texture with top 4 results
1.2 Definition of the Combining Problem

object (e.g. heraldic images in HERON [KEUB+98]) a total score value is computed based on the different features. 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 query focusing on only one specific feature will in the following be called atomic. Any complex multimedia query can be seen as a combination of atomic subqueries.

A major performance bottleneck is the combination of ranked results sets for atomic queries in order to get a certain number \( k \) of overall best objects (documents, images, texts, etc.) from a collection of data sources.

1.2 Definition of the Combining Problem

To define the combining problem there has to be a number \( n \) of output streams that have been created by retrieving \( n \) ranked result sets to answer \( n \) atomic queries. For each atomic query one output stream is generated. These streams assign a score value – typically between 0.0 (no match) and 1.0 (exact match) – to each database object. For any query \( Q \) and with \( \mathcal{O} \) as the set of all multimedia objects a scoring function \( s_Q \) can be denoted as:

\[
s_Q : \mathcal{O} \rightarrow [0,1] \\
x \in \mathcal{O} \mapsto y \in [0,1]
\]

The score value shows the 'grade of match' for each object. The lower its score value the less an object matches the query. The score values are computed by means of a similarity metrics, which of course strongly depends on the kind of feature. In general the output stream for different features are statistically independent from each other. With \( x_1, \ldots, x_N \in \mathcal{O} \) and \( w_1, \ldots, w_N \in \mathcal{O} \) a permutation of \( x_1, \ldots, x_N \) a query thus delivers the result set:

<table>
<thead>
<tr>
<th>Rank</th>
<th>( Q_1 )</th>
<th>\ldots</th>
<th>( Q_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( (x_1, s_{Q_1}(x_1)) )</td>
<td>\ldots</td>
<td>( (w_1, s_{Q_n}(w_1)) )</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>N</td>
<td>( (x_N, s_{Q_1}(x_N)) )</td>
<td>\ldots</td>
<td>( (w_N, s_{Q_n}(w_N)) )</td>
</tr>
</tbody>
</table>

Since \( n \) ranked result sets have to be combined, a combining function is needed.
1 Introduction

This function expresses the way in which results of atomic queries should be combined to form the result set of a complex query. It also may include specific weights users have chosen in order to state the relative importance of different query parts. Obviously there is no such thing as a universal all-purpose combining function, but the field of application will strongly influence the selection of an adequate function. Typical examples of combining functions include weighted arithmetical/geometrical mean, maximum or minimum. With \(s_Q\) as scoring function and \(Q_1, \ldots, Q_n\) as queries for \(n\) different features the combining function can be evaluated for each object \(x\) as:

\[
F_{Q_1, \ldots, Q_n}([0, 1]^n) \rightarrow [0, 1] \\
(s_{Q_1}(x), \ldots, s_{Q_n}(x)) \rightarrow F_{Q_1, \ldots, Q_n}(s_{Q_1}(x), \ldots, s_{Q_n}(x))
\]

If it is obvious to which subqueries the scoring functions and the combining function refer, the queries \(Q_1, \ldots, Q_n\) are often omitted. Thus for any object \(x\) the scoring functions can be written as \(s_i(x), 1 \leq i \leq n\) and the combining function as \(F(s_1(x), \ldots, s_n(x))\).

The obvious approach towards combining \(n\) different streams is to get the partial score for each object from each output stream, calculate an aggregated score for each object using the combining function and order the integrated overall result set by score. However, this approach would need several linear database scans and is therefore not applicable in practice. With growing number of the database objects, it would obviously result in far too long response times. And what is more, the whole result set is not needed, as users generally only want to retrieve a few top-scored results. A major problem of more efficient combining algorithms is thus to estimate which part of each output stream from each query is needed to calculate aggregated scores for a certain number \(k\) of overall top-scored result without missing any relevant object. In the following we will focus on optimizing the combination of atomic subqueries in different environments that guarantee a correct result set and at the same time minimize the number of objects to be accessed.
2 Combination of Ranked Result Sets in Homogeneous Environments

This chapter will focus on finding an efficient solution for the combining of ranked result sets in homogeneous environments. We understand homogeneous environments for multi-feature combination as those environments in which all the objects occur in the datasource of each classifier and a global mapping schema exists such that each object from any can be mapped onto one specific object in any other datasource.

Previous significant work in this area is due to Fagin [Fag96], who gives an algorithm that does not only provide a correct set of results, but is asymptotically optimal in terms of database size with arbitrarily high probability. We aim at proving that the new algorithm presented here will not access more objects than Fagin in the worst case, but is expected to access less objects on the average. In fact, experimental results are indicating a substantial performance gain.

2.1 The Framework for Multi-Feature Optimization

In this section we will first revisit Fagin’s algorithm and then present an algorithm featuring a new test of termination.

2.1.1 Fagin’s algorithm

In 1996 Fagin presented an approach [Fag96] 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. The expected number of objects processed for a set of statistically independent subqueries is \( \frac{k}{n} N^{(1 - \frac{1}{n})} \), where \( n \) is the number of subqueries and \( N \) is the number of objects in the database. As the complexity of our algorithm will be compared to Fagin’s later, we briefly describe his approach.

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 producing an atomic output stream.

- 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.

**Fagin’s Algorithm A1 [Fag96]:** (cf. Fig. 2.1)

Suppose that we are interested in \( k \) objects with the highest aggregated score for a monotonous combining function \( F \). Given the output streams of \( n \) atomic queries sorted in descending order:

1. Collect the output of each stream, until there is a set \( L \) of at least \( k \) objects such that each stream has output all the members of \( L \).

2. For each object that **has been seen**, i.e. that has occurred in a stream’s output, do random accesses to get the object’s missing scores for each atomic query.

3. Compute the aggregated score \( S(x) := F(s_1(x), \ldots, s_n(x)) \) for each object that has been seen and output the \( k \) objects having maximum aggregated scores.

### 2.1.2 Statistical approaches

Fagin’s work has also strongly influenced statistical approaches as [CG97]. Here a different query model is presented using not only ranking expressions for query processing, but a filter condition like the WHERE-clause in SQL. During processing first the filter condition is evaluated and later on only objects satisfying the condition are ranked according to a ranking expression. For the class of applications considered above - providing ranking expressions only - a way to map ranking expressions into filter conditions is shown and again the filter conditions can be evaluated first before
2.1 The Framework for Multi-Feature Optimization

computing scores for the accessed objects and sorting them for output. A query for the top $k$ results for a ranking expression is thus transformed into a query retrieving $k$ elements having a score larger than a threshold determined by the filter condition. Unfortunately the process of determining these thresholds from a ranking expression needs some statistical information of the score’s distribution in each query stream. This information can for example be provided by Fagin’s algorithm to determine the expected number of objects to access when the top $k$ results are requested. Though experimental results show that the number of objects retrieved can be considerably smaller than in Fagin’s algorithm, the algorithm in [CG97] is only expected to access no more objects than Fagin, but especially for small $k$ often accesses even more objects.

To guarantee a correct result as well as a gain in efficiency and to remain independent of assumptions on statistical distributions in the streams, our further analysis
Combination of Ranked Result Sets in Homogeneous Environments

focuses on Fagin’s algorithm as a basis of research. The algorithm Quick-Combine
presented later will not only guarantee correct results, but its number of objects ac-
cessed will also be upper bounded by the number accessed by Fagin. And indeed
Quick-Combine will in general access far less objects as we will prove in section
2.3.3 assuming a uniform distribution of scores and as our experimental results for
not uniformly distributed scores will show.

2.1.3 A New Test of Termination and a Basic Version of
the Quick-Combine algorithm

To combine the results of atomic output streams efficiently our algorithm has to pass
all the phases of Fagin’s algorithm. But instead of completing a phase before entering
the next, we switch between the phases and thus minimize the number of necessary
random accesses 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 \( q_1, \ldots, q_n \) aggregated using a monotone combin-
ing function \( F \). Let \( x \) be an object and \( s_i(x) \) be the score of \( x \) under subquery \( q_i \).
An object occurring in the result set of subquery \( q_i \) on rank \( j \) will be denoted \( r_i(j) \).
For each subquery compute an atomic output stream consisting of pairs \( (x, s_i(x)) \) in
descending order based on score.

**Algorithm Quick-Combine (basic version):**

0. Get some first elements from every atomic output stream.

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

2. 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:

   \[
   S(o_{top}) \geq F(s_1(r_1(z_1)), \ldots, s_n(r_n(z_n)))
   \]

   where \( z_i \) is the lowest rank that has already been seen in the stream of \( q_i \).
2.1 The Framework for Multi-Feature Optimization

3. If inequality 2.1 holds, according to theorem 1 \( o_{top} \) can be returned as top object of the whole database. If inequality 2.1 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 1 with the newly seen objects.

Consider for instance sample results of our query by visual example (cf. fig. 1.1). The following example will show how to get the top-scored object of the database \( (k = 1) \) using algorithm Quick-Combine in the basic version:

<table>
<thead>
<tr>
<th>( s_1 ) : query on texture</th>
<th>( s_2 ) : query on avg. color</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank</td>
<td>1</td>
</tr>
<tr>
<td>score</td>
<td>0.96</td>
</tr>
<tr>
<td>object</td>
<td>o1</td>
</tr>
<tr>
<td>rank</td>
<td>1</td>
</tr>
<tr>
<td>score</td>
<td>0.98</td>
</tr>
<tr>
<td>object</td>
<td>o4</td>
</tr>
</tbody>
</table>

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 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:

<table>
<thead>
<tr>
<th>random accesses</th>
<th>object</th>
<th>o1</th>
<th>o4</th>
<th>o2</th>
<th>o5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>output stream</td>
<td>( s_2 )</td>
<td>( s_1 )</td>
<td>( s_2 )</td>
<td>( s_1 )</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>score</td>
<td>0.78</td>
<td>0.84</td>
<td>0.40</td>
<td>0.83</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

After every random access the new object’s aggregated score can be computed and 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:

<table>
<thead>
<tr>
<th>test of termination</th>
<th>last object seen</th>
<th>o1</th>
<th>o4</th>
<th>o2</th>
<th>o5</th>
</tr>
</thead>
<tbody>
<tr>
<td>agg. score</td>
<td>0.87</td>
<td>0.91</td>
<td>0.64</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td>( F(\text{lowest scores}) )</td>
<td>-</td>
<td>0.965</td>
<td>0.94</td>
<td>0.905</td>
<td></td>
</tr>
<tr>
<td>( o_{top} )</td>
<td>o1</td>
<td>o4</td>
<td>o4</td>
<td>o4</td>
<td></td>
</tr>
</tbody>
</table>

After accessing the fourth object the evaluation of streams can already be stopped as inequality (2.1) holds: \( 0.91 = S(o_{top}) \geq F(s_1(o2), s_2(o5)) = 0.905 \). Now \( o4 \) - the best object that has been seen - can be returned as top-scored object of the entire database. Note that none of the objects accessed has been seen in both streams.

This algorithm will, even in the case of maximum and minimum as combining functions, return the \( k \) best matches most efficiently. In this case Fagin presents two
special algorithms that are optimal in terms of efficiency, but differ from his general approach. These two special algorithms are already contained in our algorithm simply by choosing minimum or maximum as $F$. To show that the algorithm’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 2.1 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:** It is to show that no object that has not been seen can have a larger aggregated score than the top object that has been seen.

Let formula 2.1 hold, $o_{\text{top}}$ be the top-ranked object 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 monotonicity of $F$ and equation 2.1:

$$S(o) = F(s_1(o), \ldots, s_n(o)) \leq F(s_1(r_1(z_1)), \ldots, s_n(r_n(z_n))) \leq S(o_{\text{top}}).$$

2.2 Further Improvements of Quick-Combine

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 the algorithm’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.

2.2.1 Optimization 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 2.1, it is obvious that the inequality may become true:
2.2 Further Improvements of Quick-Combine

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 our algorithm Quick-Combine the first stream $q_j$ is enlarged 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 2.1 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 may 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 $o_{\text{new}}$ 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 2.1 holds by substituting $s_j(r_j(z_j))$ with $s_j(r_j(z_j + 1))$.

**Proof:** Let $o$ be a database object. As formula 2.1 holds, the object $o_{\text{max}}$ having the largest aggregated score must have been seen according to theorem 1:

$$\forall o \notin L \land o \neq o_{\text{new}} : S(o) \leq S(o_{\text{max}}).$$

That means either $o_{\text{max}} \in L$ or $o_{\text{max}} = o_{\text{new}} \notin L$.

If $o_{\text{max}} \in L$ there is nothing to show. If $o_{\text{max}} = o_{\text{new}} \notin L$, it never occurred in any of the query streams and thus its scores satisfy

$$s_j(r_j(z_j + 1)) = s_j(o_{\text{new}}) \quad \text{and} \quad s_i(r_i(z_i)) \geq s_i(o_{\text{new}}) \quad (1 \leq i \leq n).$$

As formula 2.1 holds for an element of $L$ after the substitution, we know that

$$\exists o \in L : S(o) \geq F(s_1(r_1(z_1)), \ldots, s_j(r_j(z_j + 1)), \ldots, s_n(r_n(z_n))) \geq F(s_1(o_{\text{new}}), \ldots, s_j(o_{\text{new}}), \ldots, s_n(o_{\text{new}})) = S(o_{\text{new}})$$

due to the monotonicity of $F$. Thus there is an element $o \in L$ with

$$S(o_{\text{new}}) \leq S(o). \quad \square$$
2 Combination of Ranked Result Sets in Homogeneous Environments

2.2.2 Control Flow for Evaluation of Streams

The algorithm Quick-Combine so far considers the top-ranked element of each output stream before proceeding to the next ranked elements. As algorithm 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 seen 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 2.1 hold, means less database accesses and thus results in a more efficient algorithm.

As discussed above, there are two ways to make formula 2.1 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 expansion 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...
2.2 Further Improvements of Quick-Combine

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 $\frac{\partial F}{\partial x_i}$.

To implement our control flow using the above heuristics an indicator can be used. This indicator is calculated after every stream expansion and indicates the stream that should be expanded next. An indicator for any stream $q_i$ containing more than $p$ elements can be thus calculated as follows:

$$\Delta_i = \left| \frac{\partial F}{\partial x_i} \right| \cdot \left( s_i(r_i(z_i - p)) - s_i(r_i(z_i)) \right) \quad (2.2)$$

2.2.3 Generalization for Result Sets

Now we generalize algorithm 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 $q_1, \ldots, q_n$. 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): (cf. figure 2.1, appendix)

0. Initialization: Get the first $p$ results for each subquery, where $p$ is a suitable natural number. Compute an indicator $\Delta_i$ for each query stream $q_i$ according to equation 2.2.

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 Combination of Ranked Result Sets in Homogeneous Environments

2. Calculation of aggregated scores: Compute the aggregated score

\[ S(x) = F(s_1(x), \ldots, s_n(x)) \]

for any object \( x \) that has been seen.

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:

\[
\{ x \mid S(x) \geq F(s_1(r_1(z_1)), \ldots, s_n(r_n(z_n))) \} \geq k \] (2.3)

If inequality 2.3 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 2.3 does not hold, more elements of the atomic output streams have to be expanded. Therefore get the next element of the stream having the maximum \( \Delta \) (if the maximum \( \Delta \) is reached for two or more streams any of them can be chosen randomly).

5. Second test of termination: Check if inequality 2.3 holds using the new object’s score. If the inequality holds, return the \( k \) top-scored objects.

6. Indicator computation: Calculate a new \( \Delta \) for the enlarged stream. Proceed as in step 1 with the newly seen object.

As a control mechanism the indicator \( \Delta \) 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 algorithm Quick-Combine:

Theorem 3 (Correctness of Results)
If the output streams are evaluated until inequality 2.3 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 2.3 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
2.2 Further Improvements of Quick-Combine

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

and therefore due to the monotony of \( F \) and formula 2.3:

\[
S(o) = F(s_1(o), \ldots, s_n(o)) \leq F(s_1(r_1(z_1)), \ldots, s_n(r_n(z_n)))
\]

\[
\leq F(s_1(x), \ldots, s_n(x)) = S(x). \quad \square
\]

Since – unlike A1 – algorithm Quick-Combine will run until \( k \) objects that satisfy formula 2.3 are successively found, for \( k > 1 \) in Quick-Combine the first objects can already be returned while the algorithm is still running. With \( k = 1 \) theorem 3 shows that the first found object satisfying inequality 2.3 always is the top-scored object of the whole collection. Since this also holds for larger values of \( k \), 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 pleased by the
2 Combination of Ranked Result Sets in Homogeneous Environments

first few ranks the execution of queries for large values of $k$ can be stopped during processing (for the successive output behavior see section 2.5).

2.3 Complexity Analysis of Quick-Combine

In this section we focus on the efficiency of algorithm Quick-Combine. In particular we will show that the algorithm’s complexity is upper-bounded by the complexity of algorithm A1. Moreover, we give a general geometrical interpretation of efficiency issues and present our improvement factor in the case of uniform distribution of score values.

2.3.1 Worst Case Analysis

The following theorem will show that algorithm Quick-Combine will never retrieve more objects than algorithm A1. To this end the number of distinct objects that A1 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 2.3 always holds, when all streams have been expanded at least as far 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:

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

and thus due to the monotonocity of $F$:

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

for each of the objects $o_1, \ldots, o_k$, i.e. equation 2.3 holds. \qed

2.3.2 Geometrical Interpretation of Efficiency Issues

According to [PF95], a geometrical model for combining query results could be as shown in figure 2.3 for the case $n = 2$. If each atomic subquery is mapped onto
2.3 Complexity Analysis of Quick-Combine

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, where the coordinates refer to the object’s score for the particular subquery. For example an object’s exact match in both two subqueries of figure 2.3 would be denoted as a point in the upper right corner having coordinates (1, 1), whereas an object scoring 0.7 in the first subquery and 0.5 in the second would be denoted on coordinates (0.7, 0.5).

![Figure 2.3: Combining two atomic output streams](image)

Evaluating results of an atomic subquery by ranks can be represented by moving a hyperplane orthogonal to its axis from 1 downwards to 0. Every object that is collected by the plane occurs in the appropriate feature’s output stream. The order in which they are collected can exactly be mapped onto the ranks on which they occur.

For example consider figure 2.3 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 \( i \), whose score 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 in the upper right corner containing the top-scored objects already seen in stream 1 and 2. As algorithm A1 in its first phase collects \( k \) objects that occur in stream 1 as well as in stream 2 the dark-shaded area has to be extended in either direction by extending the light-shaded areas until there are \( k \) objects in this area. Of course according to algorithm A1 the objects collected in the shaded areas all need random accesses, as they all have been seen.

Evaluations of aggregated scores by a monotonous combining function can also be represented by moving a hypersurface collecting objects while moving over the
Figure 2.4: Hyperplane for the arithmetical mean as combining function for Fagin’s algorithm A1 (left) and the algorithm Quick-Combine (right)

area. As the $k$ objects collected first should have the top aggregated scores and thus can be returned as correct retrieval result, the hypersurface should be orthogonal to the optimal direction starting at the optimal level. Consider for example the arithmetical mean as combining function in the case $n = 2$ as in figure 2.4. The hypersurface would be orthogonal to the bisector of the angle between the coordinate axes and moving from the upper right edge with coordinates $(1, 1)$ to the lower left $(0, 0)$.

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 moves over areas containing objects for which the aggregated score has already been calculated. 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 have to be calculated, might be too large. Consider for example figure 2.4 (left) showing the arithmetical mean as combining function. The hypersurface can collect any object until it reaches the lower left-hand corner of the dark-shaded area, as not only all objects in this area have been seen, but also all aggregated scores for objects in this area 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. If e.g. a uniform distribution is assumed these two triangles will contain the same number of objects as the dark-shaded area in the 2-dimensional case thus guaranteeing correctness for the first $2k$ objects, where only the first $k$ are needed. A triangle with the same area as the dark-shaded area would also guarantee the correct-
2.3 Complexity Analysis of Quick-Combine

ness of results, but would minimize the calculations of aggregated scores and the random accesses needed, cf. figure 2.4 (right). Unlike A1 the algorithm Quick-Combine only concentrates on this minimized area always taking the combining function into account and therefore needs far less object accesses. As shown in figure 2.4 (right) for the case \( n = 2 \), streams 1 and 2 would in Quick-Combine only be evaluated down to the dashed lines instead of the whole shaded area as in Fagin’s algorithm.

2.3.3 Improvement Analysis for Uniform Distributions

Throughout the literature a special case – mostly used for analytical comparisons – is the combination of statistically independent atomic output streams with uniformly distributed score values. Fagin has proven the important result that his algorithm (cf. 2.1.1) is expected to be asymptotically optimal for these statistically independent output streams. Though uniform distributions do rarely occur in practice, it can be stated that also in this case the algorithm Quick-Combine improves A1 by minimizing the number of object accesses.

For our analysis we will again use our geometrical interpretation considered above. With growing dimensions, i.e. with growing numbers of atomic subqueries to be combined, the dark-shaded area in our model evolves to a higher dimensional cuboid whose volume determines the number of objects to calculate aggregated scores for. Depending on the combining function also in higher dimensional cases this cuboid is contained by a geometrical figure guaranteeing correctness for more than \( k \) objects.

Consider the \( n \)-dimensional case where the arithmetical mean is again taken as combining function and the atomic output streams have been evaluated down to score \( s \). Then the triangles of figure 2.4 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 together with the square to \( n \)-dimensional cuboids. In particular the polyhedron \( S_{n,s} \) is formed by the set of all points on or above the combining function’s hypersurface \( \frac{1}{n} \sum_{i=1}^{n} x_i = s \). We will show in the following theorem that the general ratio between the cube’s and the polyhedron’s volume is \( \frac{n^l}{n^n} \), i.e. the cube’s volume shrinks rapidly with growing dimensions in proportion to the polyhedron’s volume.

We have to consider the ratio \( \frac{\text{Vol}(W_{n,s})}{\text{Vol}(S_{n,s})} \) with the cuboid \( W_{n,s} = \{(x_1, \ldots, x_n) \mid s \leq x_i \leq 1 \text{ for } i = 1, \ldots, n \} \) and the polyhedron \( S_{n,s} = \{(x_1, \ldots, x_n) \mid s \leq \frac{1}{n} \sum_{i=1}^{n} x_i \leq 1 \} \).

It is obvious that the ratio does not depend on \( s \) and therefore, to make things easier, we assume \( s = \frac{n-1}{n} \). Furthermore we may replace \( x_i \) by \( 1 - x_i \) exploiting the
symmetry and consider the ratio \( \frac{\text{Vol}(W_n)}{\text{Vol}(S_n)} \), where 
\( W_n = \{ (x_1, \ldots, x_n) \mid 0 \leq x_i \leq \frac{1}{n} \text{ for } i = 1, \ldots, n \} \) and 
\( S_n = \{ (x_1, \ldots, x_n) \mid 0 \leq \frac{1}{n} \sum_{i=1}^{n} x_i \leq \frac{1}{n} \} \).

**Theorem 5 (Ratio between n-dimensional Cube and Polyhedron)**

Let \( W_n \) be the n-dimensional cube in \([0, 1]^n\) with 
\( W_n = \{ (x_1, \ldots, x_n) \mid 0 \leq x_i \leq \frac{1}{n} \text{ for } i = 1, \ldots, n \} \) and \( V ol(W_n) \) be its volume. Let further \( S_n \) denote the polyhedron 
\( S_n = \{ (x_1, \ldots, x_n) \mid 0 \leq \frac{1}{n} \sum_{i=1}^{n} x_i \leq \frac{1}{n} \} \) and \( V ol(S_n) \) be its volume.

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

**Proof:** As \( W_n \) is a cube its volume is \( V ol(W_n) = \left( \frac{1}{n} \right)^n \). Obviously \( S_n \) contains \( W_n \) and due to \( V ol(S_1) = 1 \) its volume is:

\[
V ol(S_n) = \int_{0}^{\frac{1}{n}} V ol(S_{n-1}) \cdot (x_n)^{n-1} \, dx_n = V ol(S_{n-1}) \cdot \left[ \frac{1}{n} \cdot (x_n)^{n} \right]_{0}^{\frac{1}{n}} = \frac{1}{n} \cdot V ol(S_{n-1}) = \frac{1}{n} \cdot \frac{1}{n-1} \cdot V ol(S_{n-2}) = \ldots = \frac{1}{n!}
\]

Thus \( \frac{\text{Vol}(W_n)}{\text{Vol}(S_n)} = \frac{n!}{n^n} \).

To get a more precise impression of the gain of efficiency one has to compare the total number of objects accessed by Fagin’s algorithm and the algorithm Quick-Combine. Therefore the volume of the dark- and light-shaded areas (cf. figure 2.3) 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. It is obvious that increasing volumes of the cuboids correspond to more necessary objects accesses and that the ratio between different volumes is also the improvement factor for the number of objects accessed.

**Theorem 6 (Improvement Factor)**

The total number of objects accessed by the algorithm Quick-Combine using the polyhedron \( \tilde{S}_n \), which has the same volume as the cube \( W_n \) needed by Fagin’s algorithm, is \( \left( \frac{n}{n!} \right) \) times smaller than the number of objects accessed by Fagin, if the scores are uniformly distributed.

**Proof:** The total number of objects accessed is given by the volume of \( n \) cuboids that contain a cube in the upper right corner of \([0, 1]^n\), each extended in one of the \( n \) coordinate directions 1.

---

1For example in the two dimensional case (cf. figure 3) these cuboids are the two light-shaded rectangles both including the dark-shaded square.
2.4 Experimental Results Using Quick-Combine

Let \( x = 1 - s \), i.e. the length of one edge of \( A_1 \)'s cube \( W_n \), and \( l \) the length of a cuboid, then the volume of \( n \) cuboids in algorithm \( A_1 \) is \( V \text{ol}(\text{cuboids}) = nl^{n-1}x \).

The cuboids to be analysed in Quick-Combine\(^2\) are given by the smaller cube \( \tilde{W}_n \) enclosed between the upper right corner of \([0, 1]^n\) and the hyperplane defined by the polyhedron \( \tilde{S}_n \) having the same volume as \( W_n \) (cf. figure 2.4).

As the length of the cuboids as well as their number are identical, they differ only in their base, i.e. \( x \) for algorithm \( A_1 \) and \( \sqrt[n]{V \text{ol}(\tilde{W}_n)} \) for Quick-Combine. Thus the ratio between their volumes is:

\[
\frac{x}{\sqrt[n]{V \text{ol}(\tilde{W}_n)}} = \frac{x}{\sqrt[n]{\frac{x}{n!}V \text{ol}(\tilde{S}_n)}} = \frac{x}{\sqrt[n]{\frac{x}{n!}x^n}} = \frac{x}{\sqrt[n]{\frac{x}{n!}}} = \frac{1}{\sqrt[n]{n!}}
\]

since \( \tilde{S}_n \) has been chosen such that \( V \text{ol}(\tilde{S}_n) = V \text{ol}(\tilde{W}_n) = x^n \) and according to theorem 5 the volume of \( \tilde{W}_n \) satisfies \( V \text{ol}(\tilde{W}_n) = \frac{n!}{n^n} \cdot V \text{ol}(\tilde{S}_n) \). \( \square \)

The improvement factor \( \frac{n}{\sqrt[n]{n!}} \) is according to Stirling’s formula asymptotically equal to \( \frac{n}{\sqrt{2\pi n}} \) \( \frac{(n \to \infty)}{\to} e \). This means that the algorithm Quick-Combine results in an efficiency gain growing towards 2.72 with increasing values for \( n \). Table 1 shows the improvement factors for some practical values of \( n \):

<table>
<thead>
<tr>
<th>( n )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{n}{\sqrt[n]{n!}} )</td>
<td>1.41</td>
<td>1.64</td>
<td>1.81</td>
<td>1.92</td>
<td>2.01</td>
<td>2.07</td>
<td>2.13</td>
<td>2.17</td>
<td>2.21</td>
</tr>
</tbody>
</table>

Table 1: Values of \( n \) and the improvement factors for object accesses in the uniformly distributed case

As not only visual features determine a multimedia query, but also the modern types of text retrieval deliver ranked results, typical values for \( n \) are between five and ten, but may be higher especially when large collections of data are queried by experts.

2.4 Experimental Results Using Quick-Combine

To verify the above considerations we have performed a set of experiments. In this section we will introduce the system architecture of our HERON system ("Heraldry Online"), set up an experimental scenario and present the results of our experiments.

\(^2\)Here the cubes \( W_n, \tilde{W}_n \) and the polyhedron \( \tilde{S}_n \) are placed at point \((1, \ldots, 1)\) instead of \((0, \ldots, 0)\).
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.

2.4.1 Benchmark Results for Practical Data

The HERON research prototype system (which will be described more deeply in the case study in section 4.4) has been used to process a set of atomic queries on a collection of heraldic images. Besides components for advanced query composition, user specific image delivery and efficient format conversions, it features a combining engine that can use different visual retrieval systems and universal databases (cf. fig. 4.1). The combining engine implements Quick-Combine. For our experiments we used IBM DB2 V 5.2 and QBIC technology [FBF+94] for the visual retrieval part.

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 2.3 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, 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 \( n = 3 \).

**Benchmark**

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 2.5 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 practical values of $k$. For practical values of $k$ ($k \leq 25$) Quick-Combine even accesses $30$ times less objects than Fagin’s algorithm. In our example the improvement factors for higher values of $k$ decreases, because returning 250 objects means that always 10% of the entire collection are retrieved.

![Benchmark results on real data](image)

**Figure 2.5**: Benchmark results on real data

### 2.4.2 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.

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.2.6, Fig.2.7 and Fig.2.8) 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. 2.9). A score distribution of 0.1% high and medium scores and 99.9% of very low scores was generated. Here average improvement factors around 100 can be observed for $k < 25$ as Quick-Combine adopts itself to the specific distribution. The improvement for $k \geq 50$ in this case is minimal since with $N = 10000$ and $n = 3$ there are only 30
2.4 Experimental Results Using Quick-Combine

![Graph 1](image1.png)

Figure 2.8: Average number of random accesses for skewed distributions

![Graph 2](image2.png)

Figure 2.9: Average number of object accesses for very skewed distributions

objects in the database having noticeable scores. The typical behavior of information retrieval systems is to stop the retrieval, if no more relevant objects can be detected. In our case by observing the improvement factor (or –as we will see in section 2.5– by observing the runtime of the algorithm with respect to the successive output of objects) we can determine the point, when no more relevant objects are available in an output stream.

The next diagram (Fig. 2.10) 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. With the growing size of the database obviously also the size of relevant objects increases and thus larger values of $k$ have to be considered. Improvement factors around 100 in this case can be seen for $50 \leq k \leq 250$.

The last experiment (Fig. 2.11) analyzes the scalability of Quick-Combine, if a varying number $n$ of streams is combined. We combined up to 10 different streams.
Combination of Ranked Result Sets in Homogeneous Environments

Figure 2.10: Average number of object accesses for large databases

Figure 2.11: Average number of object accesses with varying number of streams to combine

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. Therefore also the improvement factors for \( n \geq 5 \) decrease, since Fagin’s algorithm obviously cannot access more objects.

2.4.3 Discussion of 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 \( \left( \frac{\sqrt{N}}{n} \right) \) 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.
2.4 Experimental Results Using Quick-Combine

**Overall performance results:**

- 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 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 for a guaranteed correctness of results. With Quick-Combine we can get both: High performance and correct results.

However, the algorithm presented here still raises several interesting questions and open problems. As the combining algorithms are based on sorted atomic output streams generated by visual retrieval systems, developing techniques to speed up their evaluation of atomic subqueries becomes a demanding problem. For example improvements of high dimensional feature indexes or processing different atomic queries in parallel will lead to an accelerated production of output streams.

Another area of research is the evaluation of combining functions that not only express the user’s perception of similarity correctly, but also uses statistical information on the output streams in order to retrieve a highly relevant set of results. This information can e.g. be the grade of discrimination between objects in atomic output streams or the database objects’ consistency of performance. Also, the need of expensive database accesses for random accesses in heterogeneous environments can further be minimized; therefore random accesses can in part be replaced with output stream evaluations by use of an advanced flow control and suitable estimations of aggregated scores.
2.5 **Successive Retrieval of Objects Using Quick-Combine**

As stated in section 2.2.3 the top scored objects can not only be returned as a bulk, but also one after another while Quick-Combine still calculates the total result set. This section will focus on the output behavior of the algorithm for successive retrieval. As also for this behavior – like for the performance measures – the distribution of scores in each streams is essential, the output behavior is compared for different distributions. The following diagrams show the successive output behavior of Quick-Combine for three kinds of distribution. In all experiments three streams generated from a database of 25000 synthetic objects had to be combined. The queries had to retrieve 250 objects successively ($k = 250$).

All of the diagrams show the total running time, when the object on rank $i$ ($0 \leq i \leq k$) is returned. The total running time here is given in percent, as we are – unlike in the efficiency considerations – not interested in absolute times, but rather want to understand the output behavior and its implications. The first diagram is for uniform score distributions, the second shows a distribution of 1 percent of objects having noticable score values (which can be considered as the real world case) and the last diagram shows a very skewed distribution with only 0.1 percent of objects having noticable score values. The diagrams show the average of several measurements with streams showing a specific distribution.

![Successive retrieval (uniform distribution, $N = 25000, n = 3, k = 250$)](image)

*Figure 2.12: Successive output for uniform score distributions*
The diagram of figure 2.12 is based on a uniform score distribution. Though the case is very rare in practice and the performance gain through Quick-Combine is rather small, it will show the typical output behavior also within parts of distributions and is thus interesting. Consider for instance skewed distributions where only a few percent of objects show very good scores and the rest of objects has scores towards zero. Though the overall performance here is much better using Quick-Combine than for uniform distributions, the high scored objects may nevertheless be uniformly distributed with respect to each other. Thus the retrieval of only high scored objects will show a behavior depicted in figure 2.12, although the overall retrieval of more objects will lead to an output behavior shown in figures 2.13 or 2.14.

In the uniformly distributed case (cf. figure 2.12) after a period of approximately 35 percent of the total runtime for stream evaluations and the control flow, the ten first top-scored objects are returned. Following this initialization phase an almost linear output of objects can be observed. The 50 top-scored database objects are returned within 60 percent, the top-scored 100 within 80 percent of the total runtime. Thus after a fixed initialization period Quick-Combine delivers all objects one after another nearly uniformly distributed over the rest of the total runtime. Successive retrieval is obviously very desirable, since users can stop the output, if they are already satisfied by the first few results. Thus they may choose a somewhat larger number of objects to return right from the beginning instead of enlarging the return set piecewise in subsequent steps.

![Successive retrieval (practical distribution, N = 25000, n = 3, k = 250)](image)

Figure 2.13: Successive output for skewed score distributions
The diagram in figure 2.13 shows a skewed distribution assigning noticable score values to one percent of the database objects. Tests with real data using the HERON database showed that this is about the typical case in image retrieval. In this case the initialization phase consumes about 80 percent of the total runtime. Obviously the algorithm has to consider many of the noticable objects before being able to guarantee a correct result set and output the top-scored objects. Since all relevant objects have been already considered, the whole set of 250 objects to return can subsequently be output in less than 20 percent of the runtime. However, note that though the rather long initialization phase implicates a decrease of advantages through successive retrieval for the user, the absolute amount of time spent for retrieval is far smaller here than in the uniformly distributed case (cf. figure 2.15).

![Successive retrieval (skewed distribution, N = 25000, n = 3, k = 250)](image)

Figure 2.14: Successive output for very skewed score distributions

The next diagram (figure 2.14) focusses on a very skewed score distribution with only 0.1 percent of the 25000 database objects showing noticable scores. Note that 0.1 percent of 25000 objects in three independent output streams will produce a total of only about 75 high scored objects leaving a (uniformly distributed with respect to each other) mass of objects showing rather low scores. The most interesting phases in this analysis of course is the behavior of the relative runtime before and after the time when the 75 high scored objects have been retrieved. Because retrieving top-scored objects from skewed distributions has proven to be by far less time consuming than retrieving those objects from uniformly distributed sets due to less accesses (cf. section 2.4.2), it is obvious that retrieving the first 75 objects has hardly any share
2.5 Successive Retrieval of Objects Using Quick-Combine

in the total runtime for the retrieval of 250 objects. After the first 75 objects have been output, a clear change of behaviour can be stated. After a short phase of 30-40 percent of the runtime where only few objects are returned but streams are heavily enlarged (comparable with a new initialization phase), the behavior resembles the uniformly distributed case from figure 2.12 for the rest of the retrieval.

Thus the occurrence of a phase with little output, but heavy stream enlargements indicates the existence of large sets of objects in the result set having similar aggregated scores, though strongly differing from the aggregated score values of the objects formerly retrieved. Such a breakpoint can be interpreted as a quality threshold, after which a definite decline of score values can be stated and an entire bulk of similar objects is returned.

![Successive retrieval (different distributions, N = 25000, n = 3, k = 250)](image)

**Figure 2.15**: Successive output of results for different score distributions

To get the diagrams for all cases into the right place regarding an absolute runtime the total number or duration of operations necessary in each case has to be compared. Thus the cases can be represented together in one diagram (cf. figure 2.15). As can easily be seen the case of uniformly distributed score distributions shows by far the worst performance, since the improvement factor towards traditional approaches in this case has been proven to be only about \( \frac{2}{7} \) (cf. section 2.3.3). The practical (lightly skewed) distribution shows a better (here improvement factors of about 30 have been measured, cf. section 2.4.1) and almost constant performance after its initialization phase, whereas the very skewed distribution shows the best performance (improvement factors of about 100, cf. section 2.4), up to a quality threshold after
2 Combination of Ranked Result Sets in Homogeneous Environments

which it – as stated before – not only behaves like the uniformly distributed case, but also shows a similar performance.
3 Stream-Combine – An Algorithm for Heterogeneous Environments

For homogeneous environments the last chapter proposed algorithms optimizing the number of objects to be accessed. This showed to be a significant step towards efficient access of multimedia database systems. But all of these algorithms are making excessive use of so-called random accesses providing the scores for a specific object in each of the result sets. These random accesses may especially in heterogeneous environments or the field of semi-structured data be extremely expensive [WHRB99]. Thus algorithms accessing more objects while preventing the need of random accesses may in special cases be more efficient than those only optimizing the number of objects to access [GGM97]. In the following section an approach is presented that does not need any random accesses while still accessing only a small number of database objects.

3.1 Related Work

General middleware like IBM’s GARLIC [CHS+95] or various kinds of Internet sources or enterprise information portals [GGM97, Coh98] try to integrate heterogeneous datasources which may also strongly differ in the type of features that are offered for retrieval. Today this task is mostly solved by middleware technology. Often a middleware solution is offered featuring a query engine for splitting up complex queries and posing them to an underlying database system that is extended by advanced content-based retrieval facilities. The Combining Engine collects all the incoming output streams and tries to find the top-scored objects using a suitable
combining engine. Then the resulting multimedia objects or documents can be retrieved from the database and delivered via the Internet. However, existing combining algorithms are designed for homogeneous environments and tend to deteriorate to complexities worse than the linear scan for heterogeneous environments [WHRB99].

In heterogeneous environments the assumption is made that for integration of results local attributes or identifiers can easily be mapped into an appropriate global domain by normalization. However, in many cases this assumption does not hold. Determining if two name constants should be considered identical can require detailed knowledge of the world, the purpose of the user’s query, or both [Coh98]. Thus for combining output streams in heterogeneous environments performing a random access would lead to comparing one object from any of the streams to all the database objects in the other streams. However, since the object is compared to all the database objects, this would mean linear database scans for every random access. Combining algorithms using only sorted accesses would break down the complexity to comparing new objects to all the objects that have occurred so far in any of the output streams. Therefore if an object is retrieved by sorted access from any stream, it has to be decided if an identical object has occurred in any other stream. If so an (artificial) global identifier can be assigned to ease the handling of the object. This global identifier is mapped to all local identifiers of the object in different data sources.

In the following we will present the algorithm Stream-Combine [GBK01], which retrieves the $k$ overall best database objects for any query consisting of $n$ subqueries without any random accesses. Nevertheless the algorithm guarantees that no relevant database object is missed. Besides, our algorithm can guarantee a correct result set and will output objects successively as soon as they are found.

The aggregated score of all objects seen by Stream-Combine can be estimated by the correct scores that have been seen and the lowest scores seen in every stream where the object has not yet occurred. This estimation is an upper bounding to the object’s aggregated score due to the sorting of the streams. Thus there are two kinds of aggregated scores:

- **Calculated scores**, that are exactly known, because the object has already occurred in every stream.

- **Estimated scores**, that are an upper bounding for the aggregated score of an object.

To guarantee the correctness of the retrieval result after the termination of our
algorithm at least $k$ objects have to be seen in every stream and no object may exist having an estimated score larger than the $k$ best calculated scores. Only in this case the $k$ best calculated scores can be returned, as all other database objects have scores upper-bounded by the smallest of the $k$ calculated scores.

### 3.2 Algorithm Stream-Combine (Basic Version)

Given $n$ atomic output streams $q_1, \ldots, q_n$, a combining function $F$ and the number $k$ of overall best results to be retrieved:

1. **Initialization**: Get one object of each atomic output stream.

2. **Initializing datastructures**: For each new object seen a tuple has to be initialized. It contains a global object identifier (oid) for the object, local oids, and scores for each stream where the object has already occurred. Local oids and the score for those streams, where the object has not been seen yet, are initialized to NULL. Thus if in the previous step an object has been seen in stream $q_i$ it has to be compared to all the objects in the streams $q_j (1 \leq j \leq n, j \neq i)$. If it proves to be an object never seen before in any stream, a new tuple and global oid has to be initialized. Otherwise the local oid and the score in stream $q_i$ is updated in the already existing tuple to which the object belongs.

3. **Updating score estimations**: As every stream expansion may decrease the estimation for the aggregated score of any partially known object, the unknown score values are set to the lowest value seen in each stream so far. These values are upper bounds the unknown scores.

4. **Calculation of aggregated scores**: For all objects with a new score estimation or new exact score the aggregated score has to be calculated.

5. **Check for final results**: If an object $o_{top}$ has been seen in all of the streams, i.e. for the object all local oids and scores are known, check if there is an object having an estimated aggregated score that is higher than the one of $o_{top}$. If there is no such object, $o_{top}$ can be output as next overall best object and its datastructure is erased. If already $k$ objects have been output the algorithm terminates.
6. **Stream expansion:** Get a new object by expanding any stream for which the local oid of the object having the maximum aggregated score is still missing and proceed with step 2.

To show that our test of the final result can guarantee a correct result set we state the following theorem:

**Theorem 7 (Correctness of Algorithm Stream-Combine)**

If an object \( o_{\text{top}} \) has a calculated score that is larger or equal the calculated or estimated scores of all objects that have been seen, it is already the top-scored object of the entire database.

**Proof:** Let \( o \) be a database object having a larger aggregated score than \( o_{\text{top}} \). Because \( o_{\text{top}} \)'s aggregated score is supposed to be larger than the exact aggregated scores respectively their upper bounds, of all objects seen, \( o \) can’t have been seen so far.

If \( o \) has not occurred in any output stream so far, i.e. it has not been seen, its score values must be less or equal the lowest scores seen in each stream so far. As the score of \( o_{\text{top}} \) has been calculated, the object has occurred in every stream. Its scores must – due to the descending sorting of the output streams – thus be larger than or than the lowest scores seen in each stream so far and due to the monotonic combining function the aggregated score of \( o_{\text{top}} \) is larger or equal than the score of \( o \), which is contrary to our assumption. \( \square \)

A short example will help us understand how Stream-Combine works and will lead to heuristics for further improvements. We will again consider a query by visual example and combine the result with a query on keywords. To answer the complex query consisting of a text query \( q_1 \) and an image query \( q_2 \) the following two atomic output streams might be retrieved. For the ease of understanding we will use the same oids for corresponding objects in heterogeneous streams and also assign them as global oids. Imagine that we are interested in the top-scored object of the entire database \( (k = 1) \) and use an equally weighted arithmetical means as combining function \( F \).

<table>
<thead>
<tr>
<th>rank</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>0.98</td>
<td>0.93</td>
<td>0.71</td>
<td>0.71</td>
<td>0.70</td>
<td>...</td>
</tr>
<tr>
<td>object</td>
<td>o4</td>
<td>o5</td>
<td>o6</td>
<td>o3</td>
<td>o7</td>
<td>...</td>
</tr>
</tbody>
</table>
3.2 Algorithm Stream-Combine (Basic Version)

We start by getting one element from each stream (step 1). The objects accessed are then stored as tuples in a suitable datastructure with the object’s global oid (step 2), its (estimated) score in both streams (step 3), the information in which stream it already has been seen and its upper bound for the aggregated score (step 4):

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.96</td>
<td>no</td>
<td>0.97</td>
</tr>
<tr>
<td>o1</td>
<td>0.98</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.97</td>
</tr>
</tbody>
</table>

As there was no object seen in both streams (step 5), we continue to collect objects alternately from each stream (step 6) and again use the lowest score seen in each stream as estimation for missing scores:

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.915</td>
</tr>
<tr>
<td>o1</td>
<td>0.71</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.835</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.89</td>
</tr>
<tr>
<td>o2</td>
<td>0.71</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.795</td>
</tr>
<tr>
<td>o6</td>
<td>0.71</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.78</td>
</tr>
<tr>
<td>o3</td>
<td>0.71</td>
<td>no</td>
<td>0.85</td>
<td>yes</td>
<td>0.78</td>
</tr>
</tbody>
</table>

The next expansion of stream 1 reveals the missing score for the already known object o3:

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.915</td>
</tr>
<tr>
<td>o1</td>
<td>0.71</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.835</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.89</td>
</tr>
<tr>
<td>o2</td>
<td>0.71</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.795</td>
</tr>
<tr>
<td>o6</td>
<td>0.71</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.78</td>
</tr>
<tr>
<td>o3</td>
<td>0.71</td>
<td>yes</td>
<td>0.85</td>
<td>yes</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Now object o3 is completely known, but unfortunately its aggregated score is the lowest of all objects. Thus it is of no interest for us and we have to go on expanding stream 2. This expansion reveals the score of object o4:
3 Stream-Combine – An Algorithm for Heterogeneous Environments

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.84</td>
<td>yes</td>
<td>0.91</td>
</tr>
<tr>
<td>o1</td>
<td>0.71</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.835</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.84</td>
<td>no</td>
<td>0.885</td>
</tr>
<tr>
<td>o2</td>
<td>0.71</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.795</td>
</tr>
<tr>
<td>o6</td>
<td>0.71</td>
<td>yes</td>
<td>0.84</td>
<td>no</td>
<td>0.775</td>
</tr>
<tr>
<td>o3</td>
<td>0.71</td>
<td>yes</td>
<td>0.85</td>
<td>yes</td>
<td>0.78</td>
</tr>
</tbody>
</table>

This time we find an object o4 whose calculated score is higher than all of the estimated scores and thus can be output as best object of the entire database, though only six database objects have been analyzed so far using eight sorted accesses.

3.3 Improvements for Stream-Combine

To improve the performance of algorithm Stream-Combine some simple heuristics are helpful. The early termination of our algorithm is essential, because less expansions of output streams mean less expensive database accesses, less tests for equality of objects from different streams and thus a more efficient algorithm. We have two major tasks during the runtime of Stream-Combine:

- We have to detect objects, whose scores are known in all of the streams, i.e. whose aggregated scores can be calculated.
- We have to eliminate objects having higher estimated scores than the highest calculated score. This can be achieved by stream expansions that either reveal the missing score values for the object or decrease its estimated score sufficiently.

3.3.1 Retrieving the Set of k Overall Best Objects

Our aim is to successively retrieve a set of $k$ overall best objects. However, as the classifiers on which the ranking is based already abstract from the object, the exact ranking amongst these $k$ objects may not be as important as the speed of deliverance. Very often performing a useful preselection of $k$ overall best objects without ranking is sufficient, leaving the final decision about an objects relevance to the expert user. The advantages of this new retrieval strategy are twofold: Obviously users can start working earlier with a part of the correct result set. If any retrieved object already
satisfies their needs, the execution of the query can be stopped. In addition, the user very quickly gets a coarse impression how the final result set will look like by seeing some representatives. Thus if it becomes clear that the result set will not be satisfying, the query can already be refined, e.g. by relevance feedback in an early stage, where only little retrieval time was wasted.

Our first improvement of the basic algorithm thus focuses on the check for final results. Note that using the new retrieval model after delivery of the $k$-th object, our algorithm still guarantees that the result set will consist of the $k$ overall best objects. Of course the $k$ objects can be ranked on demand after retrieval of the full set. If the set of $k$ objects has to be retrieved, we do not have to wait until a known object is the overall top-scored object before output. By adapting our theorem 7 we can guarantee that the first known object within the set of the $k$ highest estimated objects will always belong to the $k$ overall top-scored objects. Please note that in the final ranking it can occur on any rank from 1 to $k$, but it will always be amongst the $k$ overall best objects. For instance, in our above example we have already calculated the final score of object $o_3$ and found it to be amongst the five best objects. Thus if $k \geq 5$ had been chosen, we could already have output $o_3$.

After the first object has been delivered, we have to find the next known object amongst the top $k$ estimated objects, and so on, until we finally have to find the $k$-th object with a known score higher than every estimated score. Eventually, as $k$ objects have been retrieved, we can rank them by their aggregated scores and get a final sorting.

### 3.3.2 Optimizing the Choice of Streams for Further Expansion

To force early termination by useful stream expansions, indicators may be used to select those stream expansions that promise the best improvements regarding the above tasks. The indicator therefore has to detect:

- Streams showing distributions of rapidly decreasing score values
- Streams that are contributing most to the aggregated score
- Streams whose further expansion decreases estimated scores for a maximum number of the $k$ top-scored objects
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 estimated scores, streams showing a behavior of declining scores most rapidly relative to the ranks should be preferred for evaluation. 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.

Indicators also have to consider the importance of each stream for aggregating scores. Any decline of streams with low weights should naturally be regarded less important than declines of highly weighted streams which should get prior evaluation. As the weights are 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 $\frac{\partial F}{\partial x_i}$. The last task for our indicator is to count how many estimations of the $k$ top-scored objects can be improved by expanding a certain stream. Of course e.g. expanding a stream whose scores are already known for all objects with estimated overall scores will not reveal any useful information and thus has to be avoided. In general, the more objects can be improved by a stream expansion, the more useful information can be derived. Our indicator should thus count the number $\#M_i$ of the $k$ top-scored objects that will possibly benefit from expansion of stream $q_i$, as the respective atomic score is still missing. However, as stated before, here $\#M_i$ only needs to count the number of improvements among the first $k$ objects. After the expansion for each of those objects the atomic score will either be revealed or the estimation can be decreased.

**Indicator Computation:** With $z_i(o)$ as the score of object $o$, $r_i(x)$ as the object on rank $x$ and $z_i$ as the lowest rank stream $q_i$ has been expanded to, an indicator for any stream $q_i$ containing at least $p$ elements can be calculated as follows:
### 3.3 Improvements for Stream-Combine

\[
\Delta_i = \# M_i \cdot \left| \frac{\partial F}{\partial r_i} \right| \cdot \left( s_i(r_i(z_i - p)) - s_i(r_i(z_i)) \right) \tag{3.1}
\]

#### 3.3.3 Pruning the Search Space

After \( k \) objects have been seen in all of the streams and their aggregated scores have been calculated, due to theorem 7 no totally new object can have a higher aggregated score than the \( k \) objects. Therefore during further stream expansions only those objects that have already been seen need to be updated. In our example from above after object \( o_3 \) has occurred in all of the streams, no entirely new objects have to be collected any more. Thus no new tuples have to be initialized for those objects and they don’t have to be taken into account for further comparisons and updates.

#### 3.3.4 Algorithm Stream-Combine (Full Version)

The following full version of the Stream-Combine Algorithm was first presented in [GBK01]. It features all the above improvements.

Given \( n \) atomic output streams \( q_1, \ldots, q_n \), a combining function \( F \), the number \( k \) of overall best results to be returned and a counter \( j \) for the number of objects that still have to be output:

1. **Initialization**: Get \( p \) objects of each atomic output stream and calculate indicators according to equation 3.1 for each stream. The counter \( j \) is set to \( k \).

2. **Initializing datastructures**: For each new object seen a tuple has to be initialized. It contains an global object identifier (oid) for the object, local oids, and scores for each stream where the object has already occurred. Local oids and the score for those streams, where the object has not been seen yet are initialized to NULL. Thus if an object has been seen in the previous step in stream \( q_i \) it has to be compared to all the objects in the streams \( q_m(1 \leq m \leq n, m \neq i) \). If it proves to be an object never seen before in any stream and if less than \( j \) objects have been seen in all of the streams, a new tuple and global oid has to be initialized. Otherwise the local oid and the score in stream \( q_i \) are updated in the already existing tuple to which the object belongs.

3. **Updating score estimations**: As every stream expansion may decrease the estimation for the aggregated score of any partially known object, the unknown
3 Stream-Combine – An Algorithm for Heterogeneous Environments

score values of each object are set to the lowest value seen in each stream so far.

4. **Calculation of aggregated scores:** For all objects a new aggregated score has to be calculated/estimated.

5. **Check for final results:** If an object $o_x$ has been seen in all of the streams, check if there are $j$ or more objects having estimated aggregated scores that are higher than the one of $o_x$. If there are less objects, $o_x$ can be output as one of the top $k$ overall best results, its tuple is erased and $j$ has to be decreased by 1. If already $k$ objects have been output, i.e. $j = 0$, the algorithm terminates.

6. **Stream expansion:** Get a new object by expanding a stream having $\#M_i \neq 0$. If there are several such streams, choose one with the maximum indicator for expansion, calculate a new indicator for the expanded stream and proceed with step 2.

Consider the above example. We will again use the streams $q_1$ and $q_2$, the arithmetical means as combining function and this time will retrieve the two top-scored objects ($k = 2$). Besides, we will use our new indicator with $p = 1$ for simplicity. In order to get a smoother runtime behavior, larger values for $p$ are advisable. But in spite of rather abrupt changes in the score differences, our heuristics nevertheless work reasonably.

For our initialization phase and to calculate indicators we need two objects from each stream and get the following table (steps 1 - 4). For the indicators we know that $\frac{\partial F}{\partial x_1} = \frac{\partial F}{\partial x_2} = 0.5$ for the arithmetical means.

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.88</td>
<td>no</td>
<td>0.93</td>
</tr>
<tr>
<td>o1</td>
<td>0.93</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.945</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.88</td>
<td>no</td>
<td>0.905</td>
</tr>
<tr>
<td>o2</td>
<td>0.93</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.905</td>
</tr>
</tbody>
</table>

The score differences are:

$$s_1(r_1(2) - r_1(1)) = s_1(o4) - s_1(o5) = 0.98 - 0.93 = 0.05 \text{ and } s_2(r_2(2) - r_2(1)) = s_2(o1) - s_2(o2) = 0.96 - 0.88 = 0.08$$

For $\#M_i$ we only need to consider the first $j = 2$ top-scored objects and see that expansion of any of the streams would always help to improve only one object, i.e. $\#M_1 = \#M_2 = 1$. 

54
3.3 Improvements for Stream-Combine

The indicators now can be initialized: $\Delta_1 = \#M_1 \cdot \frac{\partial F}{\partial r_1} \cdot (s_1(r_1(1)) - s_1(r_1(2))) = 1 \cdot 0.5 \cdot 0.05 = 0.025$ and $\Delta_2 = \#M_2 \cdot \frac{\partial F}{\partial r_2} \cdot (s_2(r_2(1)) - s_2(r_2(2))) = 1 \cdot 0.5 \cdot 0.08 = 0.04$

No object is entirely known by now (step 5). Thus we have to expand our second stream (step 6) and get:

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.915</td>
</tr>
<tr>
<td>o1</td>
<td>0.93</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.945</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.89</td>
</tr>
<tr>
<td>o2</td>
<td>0.93</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.905</td>
</tr>
<tr>
<td>o3</td>
<td>0.93</td>
<td>no</td>
<td>0.85</td>
<td>yes</td>
<td>0.89</td>
</tr>
</tbody>
</table>

Now we have to calculate a new indicator for the expanded stream with $s_2(r_2(3)) - s_2(r_2(2)) = s_2(o2) - s_2(o3) = 0.88 - 0.85 = 0.03$. As o4 and o1 are still the top-scored objects, $\#M_1 = \#M_2 = 1$. Thus $\Delta_2 = 0.015$ and we have to expand our first stream next:

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o4</td>
<td>0.98</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.915</td>
</tr>
<tr>
<td>o1</td>
<td>0.71</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.835</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.89</td>
</tr>
<tr>
<td>o2</td>
<td>0.71</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.795</td>
</tr>
<tr>
<td>o3</td>
<td>0.71</td>
<td>no</td>
<td>0.85</td>
<td>yes</td>
<td>0.78</td>
</tr>
<tr>
<td>o6</td>
<td>0.71</td>
<td>yes</td>
<td>0.85</td>
<td>no</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Now the two top-scored objects are o4 and o5 and $\#M_1 = 0$ and $\#M_2 = 2$. This means that expanding stream 1 will not lead to relevant improvements, thus according to our indicator we will expand stream 2:

<table>
<thead>
<tr>
<th>object</th>
<th>$s_1$</th>
<th>seen in $s_1$</th>
<th>$s_2$</th>
<th>seen in $s_2$</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>o4</strong></td>
<td><strong>0.98</strong></td>
<td>yes</td>
<td><strong>0.84</strong></td>
<td>yes</td>
<td><strong>0.91</strong></td>
</tr>
<tr>
<td>o1</td>
<td>0.71</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.835</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.84</td>
<td>no</td>
<td>0.885</td>
</tr>
<tr>
<td>o2</td>
<td>0.71</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.795</td>
</tr>
<tr>
<td>o3</td>
<td>0.71</td>
<td>no</td>
<td>0.85</td>
<td>yes</td>
<td>0.78</td>
</tr>
<tr>
<td>o6</td>
<td>0.71</td>
<td>yes</td>
<td>0.84</td>
<td>no</td>
<td>0.775</td>
</tr>
</tbody>
</table>

Object o4 is indeed the overall best object and its tuple can be removed and output. Our output counter $j$ is set to 1 and we have to improve the top-scored object o5,
hence \( \#M_1 = 0 \) and \( \#M_2 = 1 \) and we have to expand stream 2:

<table>
<thead>
<tr>
<th>object</th>
<th>( s_1 )</th>
<th>seen in ( s_1 )</th>
<th>( s_2 )</th>
<th>seen in ( s_2 )</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>0.71</td>
<td>no</td>
<td>0.96</td>
<td>yes</td>
<td>0.835</td>
</tr>
<tr>
<td>o5</td>
<td>0.93</td>
<td>yes</td>
<td>0.83</td>
<td>yes</td>
<td>0.88</td>
</tr>
<tr>
<td>o2</td>
<td>0.71</td>
<td>no</td>
<td>0.88</td>
<td>yes</td>
<td>0.795</td>
</tr>
<tr>
<td>o3</td>
<td>0.71</td>
<td>no</td>
<td>0.85</td>
<td>yes</td>
<td>0.78</td>
</tr>
<tr>
<td>o6</td>
<td>0.71</td>
<td>yes</td>
<td>0.83</td>
<td>no</td>
<td>0.77</td>
</tr>
</tbody>
</table>

The final aggregated score of \( o_5 \) can be calculated and it is larger than all other scores. Thus \( o_5 \) is the second best object. The use of indicators has essentially improved our performance and this time with using only eight sorted accesses (as in the example for the basic algorithm) we got the two top-scored objects. The indicator especially prevents us from investing in sorted accesses that cannot provide any helpful information. Note that due to the indicator we did not have to calculate the irrelevant aggregated score of object \( O_3 \) unlike in the previous example.

### 3.3.5 Using Lower Bounds for Stream-Combine

Another improvement of our Stream-Combine algorithm can only be applied, if the exact aggregated scores for the result set is not important. Then instead of having to see the result objects in every stream lower bounds can be used. Since every object \( o_x \) has a score value between 0 and 1 with respect to each stream, not only an upper bound can be estimated for its aggregated score, but also an lower bound using \( s_i(o_x) = 0 \) as long as the exact score for stream \( i \) is unknown. Note that the estimated values for these lower bounds can only be increased, if the exact score of an objects gets known, and of course that the lower bound for aggregated scores is always smaller than or equals the upper bound. Obviously, if upper and lower bound are equal for any object we exactly know the aggregated score of the object.

In Stream-Combine we had to guarantee that the aggregated score of an object \( o_x \) is known and there is no object having a larger estimated aggregated score than the known score of \( o_x \), before outputting \( o_x \) as the top-scored object of the whole collection. However, our proof of correctness for the algorithm Stream-Combine also holds, if the termination condition is changed using lower bounds. Thus, if there is an object \( o_x \) whose lower bound for its aggregated score is larger than or equals the estimated/calculated upper bounds for the aggregated scores of all other objects in the collection, we can state that it is the top-scored object, though we still do not
know its exact score. This termination condition obviously includes the condition used in Stream-Combine, because if the exact score of $o_x$ is larger than or equals the upper bounds of all the other objects, of course also the lower bound for $o_x$ ’s score does.

Thus using lower bounds mainly has two advantages:

- If the lower bound and upper bound for any object are equal, we know its exact score value, though we may not have seen the object in every stream so far.

- The new termination condition can essentially improve the retrieval time and will never need longer than the retrieval of Stream-Combine.

It has to be admitted that the first advantage only occurs if an object’s unknown atomic score value is 0 and the respective atomic stream has already been expanded down to an object with score 0. And –as mentioned above– the second advantage is only of use, if the exact score does not have to be known for the further processing of the top-scored object. However, especially for skewed distributions or to find outliers most quickly considerable speed-ups can be gained using this variant of algorithm Stream-Combine. A short example will give a better understanding of the improvement. Consider two atomic streams $q_1$ and $q_2$ together with a weighted arithmetical means as combining function $F(x) = \frac{3s_1(x) + s_2(x)}{4}$, i.e. $q_1$ having the triple weight of $q_2$.

<table>
<thead>
<tr>
<th></th>
<th>$q_1$</th>
<th></th>
<th>$q_2$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rank</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>score</td>
<td>0.9</td>
<td>0.6</td>
<td>0.59</td>
<td>...</td>
</tr>
<tr>
<td>object</td>
<td>o1</td>
<td>o2</td>
<td>o3</td>
<td>...</td>
</tr>
</tbody>
</table>

With two sorted accesses on objects $o1$, $o2$ in stream $q_1$ and one sorted access on $o4$ in stream $q_2$ we can calculate lower and upper bounds. The lower bound for the aggregated score of $o1$ is $\text{lower bound}(F(o1)) = \frac{3s_1(o1)+\text{lower bound}(s_2(o1))}{4} = \frac{3\times0.9+0}{4} = 0.675$. The upper bounds for the scores of $o2$ and $o3$ can be calculated as:

\[
\text{upper bound}(F(o2)) = \frac{3s_1(o2)+\text{upper bound}(s_2(o2))}{4} = \frac{3\times0.6+0.8}{4} = 0.65 \quad \text{and also} \\
\text{upper bound}(F(o4)) = \frac{3\times\text{upper bound}(s_1(o4))+s_2(o4)}{4} = \frac{3\times0.6+0.8}{4} = 0.65.
\]

Thus we already know after three sorted accesses that $o1$ is the top-scored object without having seen it in every stream and knowing its exact score.
3.4 Efficiency Issues of Algorithm Stream-Combine

The big difference between Quick-Combine and Stream-Combine is the number of random and sorted accesses they use. Generally speaking Stream-Combine will use by far more sorted accesses, but no random accesses. Thus the algorithm Stream-Combine can be used in environments, where random accesses are impossible to perform. But a far more frequent case is that though random accesses may be used, their performance is considerably worse than the performance of sorted accesses. To get a better understanding of how expensive random accesses have to be like in order to use Stream-Combine efficiently, we will state a worst case estimation for the algorithm Stream-Combine. However, a real benchmark of the algorithm has not been performed yet. Since Stream-Combine is designed as a middleware algorithm, the problem in providing a real world benchmark is to estimate the number of objects in each streams that have to be transferred to the middleware.

3.4.1 Worst Case Estimation

To get an estimation when algorithm Stream-Combine will terminate, we will examine the retrieval of \( k \) objects and again use the last chapter’s geometrical model. As stated before, a condition for the algorithm’s termination is that \( k \) objects have been seen by sorted access in all the streams. Then these objects must have calculated scores that are higher than any estimated scores.

To get an impression of how far streams have to be expanded using Stream-Combine, a worst case estimation can be achieved by applying our geometrical model from section 2.3.2. As we have seen all \( k \) objects that have occurred in every stream can be located inside a cuboid in the upper corner of \([0, 1]^n\). Considering the hyperplane given by the combining function, any object that can be of relevance to the final result set has thus to be on the same side of the hyperplane as the cuboid. As the hyperplane intersects all axes parallel to those spanning the feature space, the projection of the insections onto the feature axes (where a negative value is considered as enlarging the stream down to 0) gives the score value down to which the respective stream has to be expanded at the utmost.

Thus to get a worst case estimation the following \( n \) equations have to be solved. With \( F \) as the combining function:
3.4 Efficiency Issues of Algorithm Stream-Combine

\[ F(x_1, 1, ..., 1) = C \]
\[ \land F(1, x_2, 1, ..., 1) = C \]
\[ ... \]
\[ \land F(1, ..., 1, x_n) = C \]

with \( C = F(s_1(o_{\min}), ..., s_n(o_{\min})) \) where \((s_1(o_{\min}), ..., s_n(o_{\min}))\) denotes the score values of the entirely known object \(o_{\min}\), which from all the \(k\) entirely known objects has the minimum aggregated score. These equations can be solved for almost all combining functions that are used in practice, for instance weighted arithmetical means.

Consider for instance the example of a Stream-Combine result given below where the two best objects are to be returned. We have already seen two objects in each stream and want to know how far we still have to expand the streams at the utmost:

<table>
<thead>
<tr>
<th>object</th>
<th>(s_1)</th>
<th>seen in (s_1)</th>
<th>(s_2)</th>
<th>seen in (s_2)</th>
<th>max. agg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>0.92</td>
<td>yes</td>
<td>0.8</td>
<td>no</td>
<td>0.86</td>
</tr>
<tr>
<td>o4</td>
<td>0.72</td>
<td>no</td>
<td>0.98</td>
<td>yes</td>
<td>0.85</td>
</tr>
<tr>
<td>o2</td>
<td>0.9</td>
<td>yes</td>
<td>0.8</td>
<td>no</td>
<td>0.85</td>
</tr>
<tr>
<td>o5</td>
<td>0.72</td>
<td>yes</td>
<td>0.92</td>
<td>yes</td>
<td>0.82</td>
</tr>
<tr>
<td>o3</td>
<td>0.82</td>
<td>yes</td>
<td>0.8</td>
<td>yes</td>
<td>0.81</td>
</tr>
<tr>
<td>o6</td>
<td>0.72</td>
<td>no</td>
<td>0.86</td>
<td>yes</td>
<td>0.79</td>
</tr>
</tbody>
</table>

For the worst case estimation we now use our geometrical model. Thus we have to solve the equations:

\[ \frac{1}{2}(x_1 + 1) = 0.81 \]
\[ \frac{1}{2}(1 + x_2) = 0.81 \]

Having calculated \(x_1 = x_2 = 0.62\) we know now that we have to expand both streams at most down to a score value of 0.62. If none of the already seen objects (o1, o4, o2 and o6) has occurred in all streams until this score value is reached in all streams, we can output our objects o3 and o5 as the correct overall best objects.

This estimation can even be improved if we calculate how far we have to expand the streams for our already seen objects. For instance object o1 has been seen in
the first stream having a score value of 0.92. To get a better aggregated score as the
minimum entirely known object o3 it at least has to get a score of 0.7 in the second
stream due to $F(0.92, y) = 0.81 \Rightarrow y = 0.7$. Thus with respect to o1 we only need
to expand stream 2 down to 0.7 instead of 0.62. If we do these calculations also for
o4 ($F(x, 0.98) = 0.81 \Rightarrow x = 0.64$), o2 ($F(0.9, y) = 0.81 \Rightarrow y = 0.72$) and o6
($F(x, 0.86) = 0.81 \Rightarrow x = 0.76$), we know that at the utmost we have to expand the
first stream down to 0.64 and the second stream down to 0.7.
4 Applications in Visual Retrieval Systems

With the growing amount of multimedia data visual retrieval systems will form a basic component of future database technology, since a variety of applications need retrieval of image data. Most of today’s database systems are, however, restricted to simple full-text retrieval on verbal image annotations or attribute-based retrieval on image file formats or sizes. Advanced search capabilities like content-based retrieval are included in very few databases or portal applications like for instance Oracle, DB2, Informix and the IBM EIP (cf. chapter 6). Since content-based retrieval is a most advanced technique matching images on the basis of human perception, it is absolutely necessary for any image collection without verbal annotations or applications where semantic image contents can only be named by experts. For instance in the field of art work only experts are able to comment on style and artist, but everyone can spot if two specific images are similar.

However, similarity in visual systems depends on a variety of features. Thus for almost all the queries ranked result sets have to be combined. In fact the combining algorithms Quick-Combine and Stream-Combine have both been developed for the application in content-based retrieval. We will take a closer look at the area of content-based retrieval as a first area of application for combining ranked result sets. A short overview of the useful capabilities of content-based retrieval, the SQL/MM Standard, visual retrieval systems and a case study of a digital library application - called the HERON-project - is given in the next sections.
4 Applications in Visual Retrieval Systems

4.1 Content-based Retrieval

In contrast to standard databases, text-based retrieval in image databases has proven not to be useful to the same extent. Describing the image content merely in verbal annotations which are then used for retrieval has to be considered very subjective, incomplete, error-prone and extremely cost-expensive. Another drawback is that the later use of the database has to be foreseen and thus is restricted by the annotations, as annotations for all areas of application for the image material cannot be provided. And what is more, as often only expensive application experts are able to provide correct and nearly exhaustive descriptions, a later common user with only a limited amount of application terminology may even not be able to use the database for the foreseen application anyway. In most applications, however, the similarity of images can provide useful information for any user, since even non-expert users are able to pick most similar images out of a collection.

Using these preliminaries, a major problem in image archives and databases can be stated as finding images out of a large collection which are somehow similar to a fixed image or show certain similar characteristics. To solve this problem three questions are to be answered:

- On what grounds images are considered similar by the human perception?
- How can information on these characteristics be extracted from images and how can they be represented for later comparisons?
- What measure should be used to compare these representants and how is the degree of similarity between two images described?

The answer to the first question will show which characteristics can be used to distinguish between images and which should be evaluated to order them by similarity. For this purpose cognitive psychology has studied the human perception and revealed that similarity can almost always be stated on grounds of a few simple graphical properties of an image (e.g. [TMY78]). The main issue of content-based retrieval is therefore the description of images merely based on graphical properties, which are more or less considered similar by all users. These basic properties, often called image features, include:

- Colors (average color or color distributions)
- Textures (surface structures as shadings or any periodic pattern)
4.1 Content-based Retrieval

- Shapes (contours or regions of depicted objects)
- Composition (distances between image objects, position of colored regions or layout of the image)

The second question is difficult to answer and strongly depends on the application purpose. Representations for characteristics easy to distinguish in one application area may prove particularly problematic in other areas. There are lots of different representations for the same image feature. The way of representing image characteristics can be divided into two main groups: Low-level features, where characteristics are described by few basic measures for each image (e.g. certain amounts of colors, contrasts, coarseness of image objects, roundness or enclosed areas of object contours, etc.), and high-level features, where measures become more complex (e.g. wavelet transform or neural algorithms). Low-level features are easier to extract and handle, whereas high-level features often provide more intuitive results. Of course these features can also be combined, and the selection of an appropriate set of features is the most important step in building an effective retrieval system. Generally the measures for each feature are summarized in a vector which collects all the information for a certain aspect of the image content. For example the color aspect of every image can be represented by a histogram, where a specific column is assigned to each color, whose height indicates the (normalized) amount of the color in the image. As a thumbrule it can be stated that the more complex the aspect and the more adequate it is described, the more components the respective feature vector will have.

The answer to our third question is needed for the comparison of image aspects. Once the feature vectors have been calculated an adequate metric has to be provided. For a certain aspect each image can be shown as a point in the vector space spanned by basic feature vectors for the aspect. The similarity between two images can then be calculated as the distance between the two points and \( k \)-nearest-neighbor-searches will provide the \( k \) most similar images with respect to the image content. If all components of the feature vector are independent from each other, then for instance the simple euclidean distance can be used to calculate similarity. However, in general similarity metrics have proven to be far more complex. Considering the color histograms as feature vectors of our example above, the similarity metric has not only to express, that e.g. the color orange is different from red, but also that it is nevertheless regarded much more similar to red as for instance blue (so-called crosstalk). To express this behavior complex similarity matrices have to be used and a distance (often
normalized to the interval $[0, 1]^n$, where 1 means an exact match and 0 means no similarity at all) is calculated by matrix-multiplications. As these calculations may be quite complex for high-dimensional feature vectors, a trade-off between adequacy needed and manageability of the vectors and metrics can be stated.

Since the respective retrieval technique, e.g. query by visual example or query by sketch [HK92], differs from traditional approaches, so does the form of the output. Content-based retrieval will generally not return a set of exact matches, but rather deliver a sorted list of database objects ranked by their degree of match with the query posed. Though the features to extract may have been chosen carefully, the computer aided evaluation of graphical image features may still differ from the human perception. When it comes to, for instance, color recognition a vast majority will consider orange more similar to red than to blue, but the decision will not be as clear, if purple has to be compared. Whereas some not relevant objects in retrieval results (bad precision) can be accepted and eliminated easily and quickly by users, there is no way to find missing relevant objects (bad recall). Thus the returning of (rather large) ranked result sets seems the only appropriate solution.

### 4.2 The SQL/MM Standard

Together with the development of visual retrieval engines and multimedia databases an extension of the database query language SQL towards multimedia applications has been proposed [Sto00, Mel99]. The standard proposes datatypes as well as manipulation and search facilities for multimedia specific extensions like text (part 2), spatial data (part 3), images (part 5) or facilities for data mining (part 6). For content-based retrieval especially SQL/MM Part 5: Still Image [Cot99] is of major interest. Though it does not standardize the still images and their formats, it standardizes the storing, retrieval and search of still images using SQL. Thus it is particularly useful for handling still images together with other data in database systems. Currently [Cot99] is a Final Committee Draft, but is expected to get an international standard in near future.

#### 4.2.1 A Datatype for Still Images

SQL/MM Part 5 provides an adequate datatype SIStillImage to handle image data. This datatype consists of several attributes that do not represent all information about
the image, but provide a container for basic data. Five attributes have been chosen for 
SI\textunderscore StillImage. The first two focus on the raw image data, the last three – also called 
inherent image characteristics – describe type and size of the image.

- SI\_content
- SI\_contentLength
- SI\_format
- SI\_height
- SI\_width

The SI\_content attribute contains the location where the image data is stored. This 
information may include the sample (or pixel) information, headers for the specific 
image format and all other information provided by the image as stored in the image 
file. The total size of this content is given by the SI\_contentLength attribute. The 
content length here also refers to all image information as it is stored in an image file.

The SI\_format attribute only contains the format indicator for an image. Since 
there are far too many different image formats, providing an adequate type hierarchy 
for all formats or storing all possible attributes of each format is almost impossible. 
The formats will not only differ in type, but also in versions (e.g. various JPG- 
or GIF-formats, Windows or OS/2 Bitmaps). The resulting complex type hierarchy 
would be both difficult to handle and expensive to maintain. However, by using a 
format indicator applications should be able to derive all image characteristics of 
the image content for any supported image format. For all unsupported formats the 
format indicator is set to NULL, unless the user creates a user-supplied format.

The last two attributes SI\_height and SI\_width define the size of the image. However, for some image format they might not be applicable. Consider for instance 
non-pixel-based formats (e.g. vector images), that can be scaled easily to any size 
required. But as resolutions or file sizes may be essential in retrieving images, the 
attributes have been added to the standard data type SI\_StillImage.

### 4.2.2 Methods for Manipulating Images in SQL/MM

The two manipulation tasks of still images are used for

- thumbnail generation
4 Applications in Visual Retrieval Systems

- format conversions

There are two methods `SI_Thumbnail` in SQL/MM for the creation of thumbnail images from any still image. Although thumbnails are often used in image retrieval, the thumbnail itself was not chosen to be an attribute of the `StillImage` type. This is because though obviously thumbnails are still images, they could not be stored using the `StillImage` type, since SQL does not allow self-referencing types. If stored as an attribute of type `StillImage`, the thumbnail would have a thumbnail of itself and so on.

Also the size of a thumbnail may strongly depend on the area of application and the image from which a thumbnail has been generated. If a fixed size of for instance 40 by 60 pixels for each thumbnail is implemented, the thumbnail generation from 50 by 50 images would of course be possible. But it may not be sensible, as of course thumbnails are expected to be of smaller size as the original image. Thus two methods are used to create a thumbnail of `StillImage` type from any still image. The first creates a thumbnail of a fixed, implementation dependent size, the second method accepts parameters to define width and height of the resulting thumbnail. Thumbnails are always defined to be of the same image format as the image they are derived from. If a differing format should be needed either the original image can be converted before or the thumbnail image can be converted after the thumbnail generation.

For the purpose of converting any still image into a different image format the method `SI_changeFormat` is provided. Of course due to the large amount of different formats and the area of application, any implementation will allow conversion only for a set of supported image formats. Supported formats here means that images of the types involved can be read (source format) and written (target format). After conversion, the image has to be encoded in the new format including all header information and the inherent image characteristics of the new format have to be extractable from the raw image data.

### 4.2.3 Searching Images with SQL/MM

The possibilities of retrieving images based on their content (as described in the previous section) are exactly defined in the SQL/MM standard. For image searches low-level features have to be used. The standard only implements four different features:
4.2 The SQL/MM Standard

- average color
- histogram color
- positional color
- texture

The basic feature `SI.AverageColor` sums up the color values of all the pixels in the image and divides the value by their number. For instance in RGB-images the respective intensity of red, green and blue are summed up and divided by the total number of pixels in the image resulting in an three-dimensional vector containing the average RGB-color of the image. It should be noticed that the average color of an image may not even occur in the image itself. Consider for instance an image showing roughly the same amount of red and blue color only. Though there are only red and blue in the image, the average color will be purple.

`SI.ColorHistogram` measures the relative frequency in which each color occurs in an image. Due to the coarse human color perception, similar colors can be clustered and the color space can thus be divided disjunctively into different buckets, for each of which the height of a respective column in the histogram represents how frequent a color occurred in the image.

The third feature `SI.PositionalColor` focuses on the layout of the image. Therefore an image is divided into a set of $n \times m$ rectangles, where $n$ and $m$ depend on the size of the image and the implementation. For each rectangle the average color value is calculated and thus the color layout of images can be compared region by region.

The last feature `SI.Texture` is derived from three properties that characterize the texture of an image: Contrast, coarseness and directionality. The contrast value is used to distinguish e.g. bright images with dark shadows from images showing smooth changes between colors. The coarseness of an image is defined by the size of repeating items in the image (e.g. bricks in a wall and pebbles on a beach) and the directionality value shows if there is a predominant direction in the image (e.g. leaves of grass).

For all features involving color any suitable color space (e.g. RGB, CMYK, HSV) can be chosen by the implementation as the color values are always encapsulated in the type `SI.Color`. Though the RGB color space is considered default, other color spaces can be easily adapted by providing an adequate constructor for the color space.
To compare images by content the features have to be extracted from each image and the method $SI_{\text{Score}}$ will return a score value for each image. This score value indicates the similarity of the images and can be used to rank images in the query result set. However, the absolute values of the scores do not allow direct ratings of the result’s quality. A concrete score value of e.g. 0.5 will have no exact meaning for all kind of applications and image collections. A similarity threshold, on which can be decided whether two images are considered similar, has always to be determined for each application by the user.

Another important step in retrieving images by content is the combination of any of the four basic features. Different areas of application may demand different combinations of features and/or even different combining functions. For this task a composed feature called $SI_{\text{FeatureList}}$ is provided in SQL/MM. It allows retrieval by any weighted combination of the basic features. The specific calculation of the combined result sets and the types of supported combining functions, however, is left to the implementation.

### 4.3 Commercial Content-based Retrieval Systems and Research Prototypes

Up to now there are a lot of literature surveys about multimedia databases or image archives and the need for content-based retrieval. Also different types of features, algorithms for feature extraction, matching and image indexing or complex database architectures have been proposed. But few of these publications describe prototypical implementations and only a fraction of them have reached commercial status. Most of the existing systems implement low-level features and some make also use of textual annotations in addition to the images themselves. A survey of existent systems and notable researches in the field of content-based retrieval of still images is given in [PCD+99]. As the market had no extensive use for these retrieval engines, some have already been withdrawn from the market and there is a strong tendency to integrate other multimedia datatypes as video or audio into the still existing systems. In the following sections we will have a brief overview on systems currently in the marketplace and notable research prototypes.
4.3 Commercial Content-based Retrieval Systems and Research Prototypes

4.3.1 Commercial Products

IBM’s Query By Image Content system (QBIC) [FBF+94] was build from a research prototype at IBM Almaden. It has been implemented as stand-alone visual retrieval engine and as extention for IBM’s DB2 database system (Relational Extender Image). Besides the retrieval part, there are advanced facilities for management of image data (together with different kinds of multimedia data) such as import into and export from a database, format conversions, generation, browsing and display of thumbnails and full images. The Relational Extender even allows to use image data types and their attributes in SQL queries and manipulate images with built-in functions.

However, in neither form all features of the research prototype are implemented. Although the research prototype made use of simple shape features, the QBIC system supplies only the following low-level features for image retrieval:

- average colors
- color histograms
- color layout (called ‘draw’)
- texture

Additionally content based queries can be combined with text and keyword predicates saved together with the image data and attributes like size, format, height and width or number of colors in a database.

Virage’s VIR Image Engine [BFG+96] is a set of libraries for analyzing and comparing the visual content of images and has also been implemented as stand-alone engine and as extension for Oracle, Informix, Sybase, Object Design and Objectivity databases. Virage’s VIR allows format conversions, thumbnail generation and supports the manipulation and handling of images. The stand-alone engine can even be extended by building in new types of query interface, or additional customized modules to process specialized collections of images. Besides full-text retrieval on image annotations, several default features for content-based retrieval can be used:

- color histograms
- color layout (called ‘composition’)

69
4 Applications in Visual Retrieval Systems

- structure
- texture

**Excalibur’s Visual RetrievalWare** [Inf97] is a commercial stand-alone visual retrieval engine, but has also been adapted for the use in databases as Informix. It provides feature extraction, analyzing, indexing, and retrieving of digital images based on high level features. Excalibur also offers a software development kit, that supplies enhancements and extensions to the feature extractors which provide several of the core image matching algorithms. For feature extraction the so-called Adaptive Pattern Recognition Processing (APRP) of Excalibur uses neural algorithms to prepare images for content-based retrieval. The features provided are

- color
- color layout (called 'gestalt’)
- shape
- texture

Though shape features are provided in the retrieval engine, Excalibur does not segment single objects and index their shape for retrieval, but rather partitions the image into several regions and evaluates the shape content. Therefore the shape features are suited for comparing the main composition of two or more images (e.g. face recognition). However, retrieving specific shapes from an image will in a majority of cases not lead to a satisfying result.

4.3.2 Research Prototypes

**Photobook** of the Massachusetts Institute of Technology (MIT) Media Lab [PPS94] is a research image database query environment which supports content-based retrieval. Unlike the low-level feature approach, it aims to calculate information-preserving features, from which all essential aspects of the original image can in theory be reconstructed. The features have no fixed implementation using a choice of characteristics, but the user can choose from a large number of extraction and matching algorithms and models for such queries. Basically Photobook provides a large set of algorithms for the features color, texture and shape. Photobook even contains an algorithm, called FourEyes, which incorporates relevance feedback into the other
4.3 Commercial Content-based Retrieval Systems and Research Prototypes

algorithms and allows them to integrate the user’s concept of image similarity. Together with FourEyes Photobook also can be used for interactive scene segmentation and annotation.

**VisualSEEK** The ADVENT research group at Columbia University, New York, presented the VisualSEEK retrieval system [SC97] together with a family of similar, but more specialized systems. Besides text-based searches it offers retrieval by color layout, shape and spatial relations. The queries can be posed by placing colored regions of specific shapes in absolute or relative positions in the image. Parts of this prototypical systems have been used to build versions for specific purposes. For instance to search for images in the WorldWideWeb the WebSEEK system uses color histograms and keywords from the related text.

In the **Piction** project of CEDAR [Sri95] the interaction of textual and photographic information in image understanding is explored. It presents a computational model where textual captions are used as collateral information in the interpretation of the corresponding photographs. The combination of picture and captions have been successfully applied to face recognition. A key component of the system is the utilisation of spatial and characteristic constraints (derived from the caption) in labelling face candidates (generated by a face locator). However, a direct access to the image features is currently not implemented.

**Chabot** of the University of California in Berkeley [OS95] provides a combination of text-based and color-based access to a collection of digitized photographs held by California’s Department of Water Resources. The system has now been renamed Cypress, and incorporated within the Berkeley Digital Library project at the University of California at Berkeley (UCB). However the content-based features have in the meantime been withdrawn. The work on content-based retrieval has been continued by the Blobworld project, which used colors to split images into several regions. The shape and position of these colored regions can then be compared.

At the Los Alamos National Laboratory the **Comparison Algorithm for Navigating Digital Image Databases (CANDID)** [KC94] was developed offering the features like local color, texture or shape for content-based retrieval. From these features a probability density function is computed that describes the distribution of the features. This probability density function is provided as a content signature for each image and can be matched. CANDID has been applied especially to the fields of medical imaging and satellite images.

The **NETRA** system from University of California at Santa Barbara [MM97] uses
color, texture, shape and spatial relations to provide region-based searching based on
local image properties. Interesting is its use of textural analysis for image segmenta-
tion to derive adequate shape features. Segmenting by texture has proven in NETRA
to be far more effective than the traditional color-based thresholding techniques.

The MARS project by the University of Illinois [Hua96] is like Photobook an ap-
proach that does not use a fixed set of characteristic features and similarity measures,
but rather relies on relevance feedback by the user. Thus the features and measures
are adapted to both the user’s perception of similarity and different types of im-
age collections. For the adaptation of features either the weighting of features can be
changed or single features can be replaced by others capturing the user’s intention
more adequately.

4.4 Case Study: Application of Content-based
Retrieval in the HERON-project

The success of content-based retrieval to enable databases for handling digital images
is accompanied by a variety of commercial products for virtually all popular database
engines. For example IBM’s QBIC and the Relational Image Extender for DB2, the
Virage Visual Cartridge for Oracle and the Excalibur Image DataBlade for Informix
databases have taken their place in the market. But content-based retrieval has also
generally proven to be useful in dealing with large visual information sources like
the Internet and can be found e.g. in web search engines like Altavista, as well as
in smaller, more specific web portals, e.g. collections of paintings or libraries of
ornaments.

In the framework of the interdisciplinary HERON project a historical collection
of heraldic images is made accessible to both art-historical experts and the interested
public. Taking a closer look at the typical user profile in heraldic applications, it
seems clear that content-based retrieval together with text-based search is an impor-
tant and suitable technique in this case. Though images are essential in almost all
art-historic and historic sciences, their use in heraldry is exceptional. When it comes
to the identification of particular persons or personal possessions, results can often
only be achieved by the means of heraldry.

In order to serve as an art-historical working place and Internet portal, the proto-
typical HERON-system [KEUB+98] (cf. figure 4.1) was designed using a three-tier
architecture integrated into the world wide web (WWW). In addition, specific needs of future online users have been taken into consideration such as the format and quality or resolution of each image that is requested by the clients. The storage and the delivery of all images can even be optimized in HERON based on user profiles and typical workflows. Since, on the other hand, image collections may reside in different databases or file servers and the structure of collections may strongly differ, the problems of combining different ranked result lists had also to be dealt with. For this task HERON features an own combining engine dedicated to efficiently integrating result sets.

4.4.1 Middleware-Architecture of the HERON-System

For building a prototypical system within the research for the arthistorical working place a homogeneous database environment has been chosen. As database IBM DB2 V5.2 has been used extended by the DB2 Relational Image and Text Extenders for the visual and fulltext retrieval part. Since HERON was build to research middleware technology for image archives, an essential decision for the design was to focus on componentware technology using commercial products together with own developments.

Between the server with commercial database and retrieval components and graphical user interface on the client side figure 4.1 shows three middleware engines. The query engine is used to distribute complex multimedia queries which can contain both fulltext parts and content-based queries or sample images. It may also pass information about user specific image formats or resolutions needed to the multimedia delivery engine. The combining engine collects the (ranked) result sets from the different retrieval components and efficiently calculates a set of overall top-scored objects. This set is handed on to the multimedia delivery engine that may convert images to any format or resolution and passes the final result to the user. The delivery engine will also periodically optimize the image database and dynamically decide in which formats or resolutions the images should be stored to get the optimal ratio for storage size and conversion times.

All user interfaces on the client side are implemented in Java to guarantee platform independence for a variety of art-historical users. In general content-based queries can be posed by providing visual examples ("Search the database for images showing a texture like this.") or by providing exact values for features like color (average RGB-values, color histograms, etc.). Besides color pickers, a special catalogue
of sample textures and layout for the use in heraldry is offered. The text features offer fulltext retrieval capabilities including wildcards, negation, synonyms and phonetic search. Another way of text retrieval is the image thesaurus relating sample figures with correct heraldic terms to supply an effective interface even for non-experts in the application domain.

**The Query Engine**

As mentioned above the query engine divides complex queries into atomic parts and passes each part to a particular underlying retrieval system adequate for its processing. A user’s query often consists of various parts as sample images for visual retrieval, verbal expressions for text-retrieval or specifications of attributes, as e.g. image formats, filesizes or date of acquirement. However, beside managing the data necessary for evaluating the atomic queries, the query engine has to pass on information necessary to evaluate the compound query. As the integration of atomic result sets is quite a non-trivial task, these information have to be passed to the combining
engine for further analysis. Among the necessary data are

- the number of objects to return
- the choice of an adequate combining function
- weights for each atomic query part

By default in the HERON combining engine the top ten objects are returned and an equally weighted arithmetical mean is used as combining function. Though there are some default values provided, neither the user’s query intension, nor the application area can be anticipated, as the HERON middleware was designed to be open for a variety of applications. Thus the user should be able to change the combining function and assign weights to particular query parts.

Another important task of the query engine is to synchronize the underlying retrieval systems and start the integration of atomic retrieval results, when all result sets have been computed.

**The Combining Engine**

The aim of HERON’s combining engine is to integrate several ranked result sets from underlying retrieval systems into a single result set containing a certain number $k$ of the overall top-scored objects. Since the complex query has been split by the query engine into $n$ parts, the combining engine first has to collect data from $n$ ranked output streams.

The HERON-system uses the *Quick-Combine* algorithm to merge atomic ranked result sets. To perform random accesses, it therefore has to satisfy the condition that all the heraldic images exist in all of the (possibly different) data sources from which the different features are extracted. However, this restriction is not a major drawback dealing with content-based retrieval in digital libraries. Furthermore HERON uses a set of global object identifiers to map the same images in different data sources onto each other. Thus in a complex query for instance the color features of IBM’s QBIC retrieval engine could be used together with the texture features of Excalibur’s Image DataBlade.

**The Multimedia Delivery Engine**

The multimedia delivery engine delivers images or – in general – multimedia data to the user. However, users may strongly differ in their specific needs and profiles. Both
current users and uses, and anticipated future users and uses, have to be considered when setting up and maintaining a data collection. The number of users and the types of access profiles must be anticipated, the location of users must be considered and the technological capabilities of user hardware must be taken into account. Providing access to multimedia or image archives using personalized GUIs or Internet browsers requires storage of a variety of formats.

In addition multimedia databases must support various types of users from extremely heterogeneous environments at different stages of work:

- Small, low-resolution browse or thumbnail images may be shown quickly to enable the viewer to identify a work of art, or review large collections.
- A medium-resolution full-screen image may accomplish a full catalog record and may be sufficient for analysis.
- A high-resolution image may be available for zooming, printing or detailed study only on special request.

Besides, images in a particular format may be sufficient for classroom use by e.g. undergraduate students, but contain too little information for a conservator exploring the technical construction of a work, for instance a 256-color GIF image may be adequate for recognition purposes, but too inaccurate to support comparative analysis of an artist’s palette. A sample user interaction with the HERON image database is shown in figure 4.2.

However, the image data stored in and delivered by multimedia database systems is often redundant, because storage formats are typically interrelated by conversion tools. They may only differ in aspects such as compression, color depth and resolution. Especially in the field of large image archives, where system architectures and network topologies become significant concerns, care must be taken in identifying the file formats to be stored and delivered.

The approach in HERON [WHK99, WHK00] enables the dynamic optimization of multimedia document delivery and allows the efficient and flexible application of multimedia database systems. This is done by giving the option either to physically represent a multimedia object or to compute it from others leading to a tradeoff between storage and conversion time. Given a set of possible conversions between multimedia objects there is generally more than one way of partitioning the formats into physically stored and those computed at runtime by means of conversions. By
4.4 Case Study: Application of Content-based Retrieval in the HERON-project

applying a cost function the database server determines its optimal choice considering specialized aspects. This optimization is an automated complex task which can be performed periodically.

4.4.2 Heraldry and Typical Query Profiles in HERON

To understand the field of application for the HERON system we have to take a closer look at heraldry. The beginnings of heraldry date back to the late eleventh century when nobles began to fight in armour, and each led his particular body of retainers into the field. As it became more and more impossible to recognize strongly armoured fighters, pictorial representations were used to identify individuals, and later on represented entire families. In the year 1095 shields emblazoned with heraldic emblems (charges) appear for the first time in history and early on, heraldry took its place as an integral part of warfare. Since the 12th century first attempts were made to lay down laws for its guidance and to collect a wide variety of different bearings of knights. These rolls of arms were the first references containing images or prose descriptions (blazonings) of bearings at various periods (cf. Figure 4.3). By the Middle Ages heraldry had blossomed into a complex system with the growing tendency
to crystallize vague guiding principles into exact rules\textsuperscript{1}.

As the acceptance of heraldry grew, the depictions of bearings – together with the refinement of arts – became more and more individual. Not only the form of the shield itself varied, but shields were decoratively surrounded by some form of drapery, helmets, crowns, crests or other badges. Nonetheless these outer ornaments are of minor interest. In general there are three principal elements that characterize a coat of arms:

- the field,
- the tinctures: metals, colors and furs,
- the charges.

The field is the ground of the shield and may be divided by horizontal, perpendicular or diagonal lines and by any combination of these. Thus smaller partitions arise that can be divided or emblazoned with charges like the original shield. Each partition or charge is of a specific tincture. The tinctures comprise two metals - gold and silver (often represented by yellow and white), seven colors - red, blue, green, purple, black, orange and brown and three furs - ermine, vair and potent. In drawings or engravings tinctures are mostly represented by dots or differently arranged lines (hatchings). There are lots of charges or symbols that can emblazon a shield, even overlaying several partitions. One of the main rules in heraldry is that metal must not be placed on metal, or color on color. For instance, if the field is of gold or silver, the charges thereon must be of color or fur. However, many exceptions are allowed and occasionally even arms violating this rule are found.

During the Middle Ages arms became a common way to identify not only noble families, but also personal possessions, and the ways to resume armorial bearings had to be restricted. Therefore heralds had to be appointed to make grants of arms and generally to observe the compliance with heraldic regulations. As heraldry spread all over Europe and the number of arms as well as charges to distinguish rapidly grew, a technical terminology had become absolutely necessary. The art of correct blazonings is a very complex matter, which requires a specific vocabulary. Not only partitions, colors and charges had to be named individually, but also particular postures and several ways of depiction.

4.4 Case Study: Application of Content-based Retrieval in the HERON-project

The interpretation of arms, that is the assignment of arms to their bearers, can help to confirm identities of persons as well as the ownership of particular objects. If an identity has already been verified correctly, e.g. names are explicitly mentioned on portraits or tombstones, the blazoning of arms painted just completes the exact description of an object. By far the more demanding case is that a coat of arms depicted is the only chance to identify particular persons.

Considering for instance the portrait shown in Figure 4.4, the identity of the person portrayed is not apparent. The only recognizable hint is given by the coat of arms painted in the upper right-hand corner, which is supposed to be the bearing of the person depicted.

Using one of the main works of reference for German heraldry\(^2\), the manual search for this bearing produced the result shown in Figure 4.3. Though the form of the shield differs, the charge and colors used are the same. Besides the illustration of the arms, there is a short text containing genealogical information as well as the blazoning. In this case the coat of arms was borne by a family named "Praun".

Having identified the arms, further analysis of provenance, time of origin, artist or comparisons to other portraits showing the same coat of arms, can verify that the bearer of the arms identified really is the person portrayed. A similar case is shown in Figure 4.5. Again there is no hint that helps to identify the portrayed person besides two different arms. A more exact analysis of the painting shows that the arms in the

upper right-hand corner was inserted about a hundred years later. Thus it is most likely to be the shield of an owner, in this case the Prauns as seen before (cf. Figure 4.3). In fact as far as this painting is concerned, neither the bearer nor the origin of the arms on the left had been ascertained by now.

Finding particular coat of arms in works of reference has been a difficult matter so far, because most works are ordered by topographic aspects, i.e. any volume only contains arms of a regionally restricted area. Furthermore there are far too many different collections of arms, preventing a complete sequential scan. For instance 'Siebmacher’s grosses Wappenbuch’ consists of more than a hundred volumes, together containing about 130,000 different arms. So it is easy to verify assumptions, but finding arms without any knowledge of their provenance or the bearer’s name is far too often - despite time consuming searches - unsuccessful. Therefore art-historical descriptions of objects frequently contain blazonings, but the identification of the arms still remains to be obtained by accident.
4.4 Case Study: Application of Content-based Retrieval in the HERON-project

4.4.3 Content-based Features for Heraldic Applications

Since there are a variety of commercial products HERON could use for the image retrieval part, the HERON-system was designed as a componentware approach to multimedia middleware. However all commercial products only support some of the features mentioned in the previous sections.

Common are the use of colors, textures and the composition of images (as defined in the SQL/MM standard), whereas the shape features are currently missing in every commercial product. Though the shapes of charges have proven to be probably the most important of all content-based retrieval features, only prototypical implementations in research systems exist. However, the problem in this case is neither finding adequate representations, nor the extraction or handling of the measures involved. But in order to extract information on contours of image objects of course all the important object contours have to be segmented. Unfortunately, up to now no algorithm is known, that allows recognizing objects in general image archives similar to the human perception [WP98]. One solution proposed has been the manual outlining

Figure 4.5: 16th Century painting showing the arms of the person portrayed (upper left corner) and the later inserted arms of a collector (upper right corner)
of contours \([ABF^* 95, BFH^* 94]\), which was shown to return suitable results \([Bal97]\), but clearly is by far too expensive for large image collections. Effective segmentation can nevertheless be provided, if the image collection deals with a specific field of application and some semantic knowledge can be exploited \([VBK00]\).

In many applications best content-based query results have been achieved using \textit{color features}. But whereas histogram color has proven to be rather useful, average color has no real application in heraldic databases, because it is impossible to distinguish clearly between shields only using their average color, e.g. a crest containing red and blue parts may have the same average color values as an all over purple one. Though comparing histograms is an effective way to distinguish between even similar shields, the relevancy of mere colors in heraldry is quite limited. Thus queries by color will only be useful in combination with additional features, such as shape or texture. Unfortunately colors cannot be used directly for querying, as most books of reference merely reproduce monochrome prints of shields, where each color is shown as a certain shading. Therefore the problem of segmentation - in particular the distinction between areas of different color - is going to become more and more important, since this is the only way to take advantage of queries by color in heraldic applications.

As main application of \textit{texture features} in heraldry the task of finding areas covered by furs can be stated. A sample query result is shown in figure 4.6. A visual example of an area covered with ermine (upper left) is compared to all images of the database. The visual example is shown in the upper left-hand corner. Since the example was taken from the database, the original image has been found first, followed by

![Figure 4.6: Query by texture](image)
4.4 Case Study: Application of Content-based Retrieval in the HERON-project

all those containing areas covered with ermine. The last image shown does not contain any ermine. Note that the distance measured for the last retrievals displayed in figure 4.8 and figure 4.6 have the same value. But the latter image is a false retrieval, whereas the former image is relevant. This shows that the meaning of absolute values for distances may strongly differ with respect to the particular feature.

Retrieval by shape features has proven to be one of the most complex problems in image retrieval. A wide variety of shape features have been proposed in machine vision literature, but similarity strongly depends on the particular field of application. The Ultimedia Manager – which in the meantime has been withdrawn from the market-place due to the segmentation problems mentioned above – was the only commercial product featuring a shape component. It determines shape features by area, circularity, eccentricity and major axis orientation as well as a set of algebraic moment invariants. In [Bal97] the retrieval of both simple geometric structures (circles, rectangles, etc.) and complex charges (lions, eagles, etc.) has been extensively studied. Though there are a lot of quite similar, but different charges in heraldry to emblazon a shield, their planarity, stylized depiction and clear recognizable borders provided most promising retrieval results and thus encourage the further use of query by shape in heraldic applications.

Of course, the quality of retrieval may differ with the sample chosen. A query result for simple geometrical shapes is shown in figure 4.7 and a result for more complex shapes is shown in figure 4.8. Obviously there are many false retrievals here. But all images of lions in the database are retrieved within a tolerable distance that is determined matching the features of each image in the database to the visual example.
As mentioned above, the major problem in shape-based retrieval is not the appropriate set of features, but rather the segmentation of object contours in image collections that has not yet been solved for the general case [ABF+95, WP98]. In order to provide an effective segmentation, the application semantics has to be exploited. For the heraldic application in HERON this means that the objects on coats of arms can differ from abstract geometrical shape to everyday’s objects as for instance stars, animals or plants. The segmentation problem in HERON can even be reduced to segmenting (shaded) objects which overlay a regularly shaded background in monochrome images, where different kinds of shadings represent particular colors. In [VBK00] this problem was addressed and a new algorithm for segmentation in heraldic collections was presented using the commercial Halcon image processing library [ES99].

### 4.4.4 Adaption of Retrieval Features to Diverse Image Archives

Like the automatic image segmentation also the specific choice of features for retrieval has to be adapted to thematically diverse image collections. Considering effective segmentation algorithms that rely on semantical knowledge of the application domain, the features have to be chosen similarly to the human perception that gains more relevant results by gathering experience in the specific domain. These experiences can subsequently be generalized with some fine-tuning to related topics and beyond.
4.4 Case Study: Application of Content-based Retrieval in the HERON-project

Consider for instance the meaning of texture in the context of heraldic applications. As shown above textures can be used to distinguish furs, but when it comes to the determination of shadings a low level texture approach as in most visual retrieval systems (cf. [FBF94], [BFG96]) will not lead to useful results. In figure 4.9 a query on texture is shown. The upper left-hand image has been used as a sample picture for query by visual example. It shows simple charges (three anchors, that also contribute mainly vertical lines) on a vertically shaded background, which is therefore the main texture. Though all images retrieved as similar show the same vertical shading in the background, the semantics of heraldry will not consider the coats of arms retrieved as high quality retrieval result. Since there are only nine colors in heraldry the number of bearings showing a red (i.e. vertically shaded in monochrome images) background will be enormous. And moreover, the result set shows only bearings with different kinds of charges on an unpartitioned shield, although the same charges on a partitioned red background or a shield featuring a partition with anchors on a red background would be far more important for a high quality retrieval result with respect to heraldry.

Obviously – as stated in the section before – the texture of heraldic images may be useful for a couple of tasks (determination of furs, extracting color features from monochrome images, etc.), but it is definitely not very helpful for retrieval. Thus a fine tuning of texture features is unnecessary, as it will not improve the retrieval result. However, texture also in this case mirrors a global perceptual similarity, because for non-experts in the field the retrieval result seems to return quite similar bearings. Thus we may expect that the adaption of the texture features chosen here will succeed for any similar field of application, where global perceptual similarity is more essential.

To evaluate the chances of generalizing the content-based retrieval used for heraldry in HERON first attempts have been made to adapt the heraldic retrieval features to somehow related art-historical domains. A promising approach was to extend the search facilities of the HERON-system towards ornaments, an application that is closely related to heraldry, but relies on more global features3. As ornaments often are spread over larger areas, they show typical periodic patterns. The content-based analysis could thus focus on texture to find and characterize these regular patterns. Similar to heraldic features the contours of image elements and textures are more

---

essential than colors as the same ornaments in different colors nevertheless evoke a strong perception of similarity.

In general ornaments can be divided into isolated and wide-spread ornaments. Whereas isolated ornaments can only be described by shape features, and the wide-spread ones can be characterized successfully by texture features. Besides the contours of image elements their relative sizes, coarseness and directionality form an adequate texture feature for this domain. All the tests focussed on area-wide patterns from ancient egyptian ornaments taken from Jones ‘Grammar of Ornaments’ that were characterized by the above texture features. Despite satisfying retrieval results, the reason for similarity between ornaments is often hard to understand and sometimes even incomprehensible.

Figure 4.9: Query result on Texture for heraldry (vertically shaded)
4.4 Case Study: Application of Content-based Retrieval in the HERON-project

Figure 4.10: Query result on texture for ornaments (stylized leaves)

For instance in figure 4.10 there are some patterns matching the query image (upper left) quite well (ranks 3-4 and 7-8). In contrast the reason for the retrieval of what is to be expected the most similar image (rank 2) is perceptually incomprehensible. However, using low-level features for retrieval will always result in the retrieval of some objects, whose relevance cannot be explained easily. But since the overall result is satisfying, false drops have to be taken into account. For an overall quality of service it is almost always far more sensible to let the user discard single false drops from any overall satisfying retrieval result than trying to further improve the result set. Although this might be possible by e.g. fine-tuning or replacing some of the features or even integrating explanatory components, it will extend the retrieval time considerably. Discarding a few false drops, however, can easily and quickly be
Figure 4.11: Query result on texture for ornaments (geometrical)

performed in a perceptional check by the end user. For example in the query shown in figure 4.11, the human perception will immediately single out false drops like the image on rank 7 and have a closer look on those images showing somehow checkered patterns like ranks 4, 6 or 12 and decide on their relevance.
The aim of this and the following chapter is to show more application areas beyond content-based retrieval for our combining algorithms (cf. [BGK00]). In the case study of the HERON project we have already seen the combining engine as a useful part of a general multimedia middleware architecture. Mainly there are three scenarios for the use of combining engines:

- **In homogeneous environments** the same set of data can be seen from different views by (possibly different) retrieval systems. Each view will rank the source’s objects individually according to their relevance. Typical examples for retrieval in homogeneous environments include content-based retrieval and multi-classifier combination.

- A second scenario are multiple datasources that consist of different data sets which have some standardized attributes in common. We will refer to these scenarios as quasi-homogeneous environments. If the common attributes are keys for the data source, objects from different sources can be grouped uniquely by sharing the same common attributes. Typical examples for retrieval in quasi-homogeneous environments include information portals. We will take a closer look at these applications in chapter 6.

- The third application are heterogeneous environments where not only the data in the various sources differs, but also no common attributes or identifiers exist. Generally there has to be a function mapping objects from different sources that belong together. These function, however, tends to have a high computational complexity; thus comparisons between objects are expensive. Typical examples for these environments are applications on semi-structured data. We
5 Applications in Multi-Classifier Combination

...have seen examples for these environments in chapter 3, but will not go into further detail as these applications still cause some severe unresolved problems and do not seem to be a common case in today’s information systems anyway.

The next sections will deal with the application of combining algorithms in multi-classifier combinations.

5.1 The Problem

The problem of multi-classifier combination is related to the problem discussed within the last chapter. But instead of ranking database objects according to different aspects of a query, in this case an object or sample is categorized. The different categories are ranked by the certainty that the sample belongs to the category. Multi-classifier approaches are often used for recognition tasks. However, since there is only a certainty that a sample belongs to a specific category, there can be considerable error rates for recognition tasks. Modern information retrieval systems try to decrease the error rate of determining the relevance of a given object to a query. However, the more precisely objects are described by the features on which decisions are based, the more their computational complexity is growing. This brings down the efficiency of the whole system. Besides, at a certain point the error often cannot be decreased any more by refining the retrieval algorithm or features. At this point it has been shown that combining the result sets of different systems, each retrieving the most relevant objects with an error rate larger than 50 percent, leads to better error rates than all of the individual systems [KR00].

The problem of multi-classifier combination is choosing a set of independent classifiers that are likely to minimize the error, calculating result sets for each classifier and fusing the different result sets to get an overall best classification for any sample.

Multi-classifier combination assumes that several classifiers perform the classification of each object individually and therefore make independent errors leading to a superior classification quality compared to each single classifier [TG96]. Since each individual classifier makes errors ‘in a different way’, it can be expected that the set of classifiers can correct the mistakes of individual classifiers. Also the efficiency of the system can be improved since several tasks of different classifiers can be executed in parallel. Since multi-classifier systems are commonly used for recognition tasks,
there are severe constraints to perform all classifications in real time. The kind and number of classifiers to combine thus strongly depends on their complexity. We will not go deeper into different kinds of classifiers or combining functions in this chapter, but rather rely on well-researched examples [KR00] and concentrate entirely on efficiency issues.

The result sets classifiers used in Multi-classifier combination can deliver are twofold. The classifier may simply return the category or set of categories to which the sample belongs or it may provide a ranked list of classes to which the sample might belong. In most cases those ranked lists are provided together with score values representing certainties, confidence levels or distances. However, for classification problems involving a large number of classes, the probability that each classifier can identify the correct class of each sample tends to decrease.

Thus in the following we will consider each classifier’s result set always as a ranked list with scores assigned. In fact, the other kind of result set can be seen as a special case of a ranked result set. In the case that a classifier only provides one or more classes we will assign a score value of 1 (exact match) to these classes and estimate scores for all the other classes with 0 (no match). The relative importance of these classifiers may be adjusted by assigning weights in the combining function.

5.2 Using Combining Algorithms for Multi-Classifier Systems

The implementation of multi-classifier combination leads to different methods and topologies [Lam00]. There are mainly three classic basic types of topologies and a variety of hybrid topologies involving combinations of the basic types:

- **Hierarchical topologies** apply classifiers in succession. Each classifier produces a reduced set of possible classes for the sample. The set is then delivered as input to the next classifier. Applications are for example the reduction of possible dictionary entries by fast classifiers such that the next (often more expensive, but more reliable) classifier may work using a (sometimes considerably) reduced dictionary.

- **Conditional topologies** apply several classifiers to the sample depending on the decision of one or more classifiers that are evaluated in a first step. If the first classifier is unable to categorize the sample with sufficiently high certainty,
the sample is passed to the next (set of) classifier(s). Generally those first classifiers are less precise, but inexpensive and fast. Only if a sample needs deeper analysis for good classification results it is passed to more reliable, but often more expensive and slow classifiers.

- **Parallel topologies** allow several classifiers to operate in parallel. The final decision then is gained by fusing the underlying classifiers’ decisions, e.g. by majority vote or any suitable combining function. A major advantage of these systems is that the operation of individual classifiers can run under parallel hardware architectures. In addition, the individual operation of classifiers allows the simple exchange of classifiers or the assignment of weights without requiring major modifications in the fusion process.

Obviously the area of application for combining algorithms like Quick-Combine or Stream-Combine lies in the fusion part of the classifier combination. Since the fusion in hierarchical topologies is made implicitly by always reducing the search space for the next classifiers, combining algorithms will not be of big help in this case. Because hierarchical topologies use strict reduction techniques the true classification of an object to recognize must always be among the subset produced by the preceding classifier, if the subsequent classification is to be correct. Therefore the use and applicability of hierarchical topologies has to be examined closely before this model is chosen.

Conditional topologies pose almost the same problem. Here also the first classifiers on which the condition is checked, have to be quite sure that their output is correct. However, combination algorithms are applicable for these topologies, although the fusion is also done only implicitly. The Stream-Combine approach can simulate the behavior of conditional topologies if high weights in the combining functions are chosen for those classifiers that should be evaluated first. Due to the high weights Stream-Combine’s indicator will chose these streams for preferred expansions and only if the highest scores in this output stream are not good enough discriminating classes the other streams are chosen for further expansions.

The typical area of application for combining algorithms like Quick-Combine or Stream-Combine, however, are parallel topologies, where all the different classifiers compute their output streams in parallel and the fusion can be done in a second step. A major advantage of this topology is that unlike in hierarchical or conditional topologies any of the systems may return a wrong classification that can be corrected by the
5.3 Case Study: Speech Recognition

The majority of the other classifiers. In both of the other topologies a misclassification of an early evaluated classifier would have a fatal effect on the final decision. Also in parallel topologies the weights of the combining function used may reflect the quality of a classification or the reliability of a single classifier. This topology is generally preferred, if all the classifiers are rather equivalent in terms of performance. Since all the classifiers have to classify the sample the final decision is often gained by majority vote, maximum, minimum, sum or means.

If the classifiers are capable of producing their classifications successively starting with the best classifications, it is even possible to start the fusion process with e.g. Stream-Combine while the classifiers are still working. The indicators in this case would have to be modified slightly, since it would not be helpful to have the algorithm waiting for the next classification of a preferred stream, while classifications from one of the other (lower rated) output streams could already be processed. The next section will focus on the application of speech recognition, where different classifiers provide result sets in parallel.

5.3 Case Study: Speech Recognition

In the area of multi-classifier combination, the evaluation of similarity between objects is based on a variety of classifiers, each of which votes for a certain ranking of the result set. In this case study we will choose natural speech recognition as an example that has been worked on very broadly. However, we will not try to solve the problem itself, but only show a way to make current approaches more efficient. As natural language has to be recognized in real time in order to communicate with e.g. information systems, efficiency is a mission critical factor. The hope for the application of multi-classifier combination is that speech recognition will provide by far a higher recognition rate, if not only phonetic features are involved, but also e.g. visual features like lip-reading or features based on the semantic context of spoken sentences [YJB00, JYB00]. Spoken words may also be assigned to certain word classes (given by an underlying grammar) with a specific certainty. As the ranking of the classifiers may differ, the overall highest certainties have to be calculated. Thus also in this case the combination of ranked result sets is necessary. Even in small grammars the resulting number of word classes has proven to be very high. The efficient combination without materialization of all the classes and certainties promises a considerable speed-up for e.g. modern speech recognition systems.
5 Applications in Multi-Classifier Combination

5.3.1 The Multi-Feature Speech Recognition Problem

Considering the problem of natural speech recognition the main approach is dividing words or sentences into classes. We will take single words as an example, but analogous considerations can be made for both phonemes or sentences and sequences of words. Starting with the acoustical or visual input different kinds of classifiers will recognize specific words from each set of phonemes with different certainties using e.g., Hidden Markov Models. These certainties can be improved by using meta-information like grammars, dictionaries or collections of possible phrases (e.g., possible combinations of three word phrases, etc.) that are often used in a postprocessing step as feedback to choose the right recognition results.

In the recognition step each classifier may rank the recognized words according to the certainty of recognition assigned as score value. At the beginning of each list there will be words appearing or sounding most similar to the input and as the score values decrease also the similarity in appearance or sound will decrease with respect to the classifier. In practice due to noise and the properties of the classifier not all of these output lists will show words in the same order. However, the chance that one of the overall top-scored words will be correct, i.e., exactly matches the spoken word, is high.

The distribution in applications like this can generally be expected to be very skewed. There will always be a small set of similar words, but words will get less similar quite soon and the majority of words in a language will not be similar to the recognized word at all, i.e., get assigned a rather small score value. Both Quick-Combine and Stream-combine feature indicators that consist of the derivative of functions correlating score values to the ranks on which they occur for each output stream and the partial derivative of the combining function for each stream. These indicators are calculated for each stream and the highest indicator determines the stream that is to be expanded next by sorted access. Due to this indicator the combining algorithms will always preferably enlarge and analyze those streams showing the most rapid decline of score values. These are the streams where the discrimination between words can be assumed to be quite high. As the algorithm is capable of working with skewed distributions a considerable speedup compared to traditional approaches may be achieved. Another advantage is that almost any number of different classifiers can be used for recognition.
5.3.2 Integration of Dictionaries within the Stream-based Approach

The recognition rate can even be increased, if e.g. a dictionary or grammar is used. The dictionary can be seen as one more output stream that consists of exact matches only. That means any word that is contained in the language or dictionary is assigned a score value of 1 and every other word will get a score value of 0. Depending on the area of application the dictionary stream should be assigned a higher or lower relevance in finding an overall best object. This can for instance be achieved by assigning high weights within the combining function.

Obviously, as there is no specific order between the words in the dictionary stream (all have a score value of 1), it will not be sensible to enlarge this stream by sorted access. However, since the indicator of Quick-Combine takes score distributions into account, any stream of that kind will be excluded from sorted accesses immediately. Quick-Combine is thus also capable of combining ranked result sets and exact match results in a very intuitive way. The lookup of for instance a word in a dictionary does not have to be a separate step in processing the input, but can be implemented by just adding another output stream containing the respective information about the language. Quick-Combine will automatically adapt its retrieval strategy to the nature of those streams containing only exact match information.

The problem of combining exact matches and ranked result sets is quite a difficult matter in modern database technology. Though today’s database systems like IBM’s DB2 are able to perform a combination within the WHERE-clause of SQL-statements, the semantic of such a combination is to filter the ranked result only taking those objects into account for the final result set that satisfy the exact match condition. Of course, by applying a condition on our combining function this filtering can also be used in Quick-Combine and Stream-Combine. However, there are applications where the exact match result set might be highly relevant though an absolute filter effect on the retrieval result might not be wanted.

Consider for instance our above dictionary example. If all the classifiers have recognized a certain word or sequence of phonemes, but it does not occur in the dictionary, the phrase may nevertheless not be wrong. It might be an neologism or very unusual phrase. In this case it would be unwise to simply ignore the unison result of all the classifiers. In the Quick-Combine and Stream-Combine approach the weighting of the dictionary stream can be individually adapted to the needs of the
5 Applications in Multi-Classifier Combination

field of application. Unlike traditional database systems our combining algorithms may integrate streams in a sophisticated way ranging from hard filter constraints to low-weighted proposals.
6 Applications in Information Portals

With the increasing use of information provided on-line via the Internet the problem of well designed access interfaces has become more and more demanding. Since today’s systems are unable to aggregate information properly, the sheer masses of information obstruct all kinds of users from getting complete and up to date information. Besides a fast and exhaustive access to various information sources, the ease of use, possibilities to personalize both interfaces and information and improving the relevance of search results are important steps towards creating an effective workplace for almost all kinds of applications. The development of so-called (Internet-)portals are the most recent approach towards advanced information access.

In general portals can be divided into two major groups. First-generation portals were those grown out of web search engines which can be considered as general purpose gateways onto the Web, like Yahoo!, Excite or Lycos. This group is often referred to as horizontal portals. A second quickly emerging group has become known as vertical portals, i.e. portals with a tightly focused content area for a specific audience. However, the area of application for portals is not restricted to the Internet, but also begins to integrate a variety of datasources from local filesystems or databases via Intranets. Especially in the field of business intelligence the workflow can be strongly improved by providing vertical portals accessing not only internal business data over various departments, but also integrating relevant information from the Internet. In the next sections we will have a short introduction to portal technology, point at some specific problems and discuss the application of combining algorithms in portal technology within a sample case study.
In recent years the amount of digital information has dramatically increased leading to a strong need of efficient ways to manage this knowledge. The first steps towards knowledge management were digital libraries and data warehouses, where a large amount of information has been made accessible efficiently for the first time. However, these applications focused on well-structured data and copied, moved or migrated data from disparate systems into a central database. Thus they were mainly based on relational databases and provided no solutions for managing personalized user profiles or handling semistructured data. It was this demand that quickly escalated the need for more advanced features which lead to the development of so-called information portals.

The main idea of an information portal is to provide a single, web-based interface to disconnected and sometimes incompatible data scattered across a variety of separate applications and repositories. As a typical example of information portals in the following we will use enterprise information portals (EIP) which had great influence on modern information systems [AGH98, Lou93, Mel94, Had90]. However, the characteristics shown for EIPs can be applied to almost all the other types of information portals.

Enterprise information portals provide one (possibly personalized) logical view and common query capabilities across different platforms and enterprise-wide information sources. In general such information sources are not limited to relational databases, image archives or document repositories, but more sophisticated portals also allow external web data to be directly integrated into query result sets [RG00]. In general the ideal goal is ‘just in time’ information, retrieved and assembled as needed, freely accessible and exchangeable across various systems, and filtered to be both manageable and usable. Thus the promises of previous approaches as intranets, data warehousing and enterprise document management are to be fulfilled by EIPs providing services for:

- administration & management
- personalization & profiling
- presentation & visualization
- categorization & metadata
6.1 A New Challenge

- extended search capabilities
- multi-repository support
- security
- workflow & process integration
- collaboration
- publishing & distribution

A major drawback in recent knowledge management approaches has been the costly overhead of administration and management of data. As all the information in EIPs is collected from well-defined registered information sources the administrative overhead can be reduced considerably. For instance logging or workspace and resource management are typical tasks that help users to organize their information flow. But the needs have grown beyond merely administrative support, especially personalization has become an absolute necessity, because the amount of information provided via Inter-/Intranet has already overstrained the processing capacity of users. The use of user profiles allows individually tailored information feeds and displays. Thus queries can be posed more intuitively, information can be adequately filtered and the results presented in a way suitable for the specific user and easy to understand. The issue of presentation is twofold: Many kinds and channels of information have to be displayed in a considerably small space and to support all kinds of users (from untrained to experts) the portal appearance has to communicate both familiarity and context. Wherever possible techniques of visualization can be used to improve the perceivability of complex data. However, the information presented has not only to be comprehensible, but also relevant and complete. Within each user community a specific terminology, domains of understanding and typical structures are prevalent creating a context that reflects the area of application. Categorization brings this information context into the portal by assigning metadata, e.g. by use of XML. Also a knowledge map, ontologies or a thesaurus of the application has to be created and reflected in the category structure of the portal.

In order to serve the user with all the information needed for a specific task portals need to aggregate information. Advanced systems therefore have to provide a variety of complex search capabilities, as e.g. full-text searches or content-based retrieval.
6 Applications in Information Portals

The typical query result will contain logical information objects dynamically created out of data distributed across heterogeneous data sources. To perform federated searches on these distributed sources a set of connectors has to be provided. Each registered information source, e.g. relational databases, file systems or web search engines, needs its own connector leading to enormous costs for both code generation and maintenance. In building connectors for multi-repository support not only the connectivity code is necessary, but connectors also have to deal with security or administration issues and a variety of operating systems, network protocols and data storage techniques. Modern EIPs therefore contain a wide variety of prebuilt connectors. When users work with a variety of content sources and applications security matters arise. Of course not all users may be allowed access to all documents. But for comfort users should be requested only once to sign on for their session. The password then has to be stored to log onto the different information sources for which the user has been granted access.

But not only the capability to access information is of major interest in EIPs, also the way of accessing and sharing information is most important. The workflow and processes of standard transactions have to be supported. Routing documents and forms, responding to intermediate changes in any business process, triggering events or monitoring instances of predefined process flows are necessary tasks towards process automation. Similarly, the importance of collaboration has to be emphasized, since the majority of today’s work is team-based. Advanced communication features, e.g. central collections of project documents, threaded discussions or chat forums, allow access to the right people in context, as well as the right content. To create content a publishing and distribution facility is needed. Authoring documents, including them in portal content collections and distributing several online or hardcopy formats are typical tasks to support collaboration and the information flow. Within the publishing facility also a life cycle management for portal content should be established ensuring the kind of up to date information users expect to receive.

The concrete architecture of any information portal is strongly dependent on the specific application area. To allow a widespread usability it has to be open to accommodate custom solutions and various third-party products. However, the general tasks any portal has to perform are threefold: Data aggregation, data integration and data access. Whereas data aggregation has to be performed on a common schema, data integration can be performed over different schemata. Some typical components EIPs generally consist of, are:
6.2 Federated Searches

- **Data aggregation:** Query templates
- **Data access:** JDBC-connectors/query portlets
- **Data integration:** Combining engines/portal metadata databases

The typical user interaction with EIPs will first provide some query information for the search together with a user profile for the later presentation of the results. The EIP will analyze the query constraints and (depending on the user’s profile and rights granted) decide which sources to query. Each source is adequately accessed and generally outputs a ranked result set assigning a grade of match to each source object. The next step is gathering the different result sets and integrating them with a combining function and weights assigned by the user to get the overall best results for the search.

Now, that we have seen portal technology as an important and necessary step towards effective use of distributed data collections, e.g. in enterprises, we will focus on the part of data integration. Data integration will provide us with the ability to gather information efficiently even from several heterogeneous data sources.

### 6.2 Federated Searches

The ability to perform enterprise-wide federated searches is one of the most advanced in EIPs. As we have seen in the case study of HERON the queries posed may not only have to return the image of a coat of arms, but also textual information like a description of the bearing, genealogical information, bibliographies or scanned images of related historic documents and research articles published in art-historical journals accessible via WWW. The same obviously is true for any kind of information portal. The answer sets will generally consist of compound documents involving data of different type collected from different sources. These compound documents contain a lot of information that has been arranged and syndicated, tailored to the specific needs of each user.

Federated searches can thus be used to get all relevant and up-to-date information as a basis for decision support, benchmarking and data mining, workflow management or even complex tasks as interdisciplinary research. For each purpose the adequate search capabilities and connectors to all sources needed have to be provided. Main types of federated searches include:
Applications in Information Portals

- Traditional full-text searches
- Fuzzy search capabilities including phonetic search or thesauri
- Attribute-based queries relying on syntactical attributes, e.g. documents of certain length or images in specific image formats
- Advanced content-based retrieval relying on semantical attributes, like semantic document retrieval or query by visual example

When an user executes a federated search in an EIP, a search template is downloaded from the portal database. The template captures user input for defining query constraints, as well as profiles for the later presentation of results, and includes cross-reference data that facilitates the search. The EIP then initiates a federated search across all the relevant content servers and adequate connector processes for each of the content servers that meet the search criteria the user specified. For all these different tasks mostly Java servlets (so-called portlets) are used [Gon00].

All data is either aggregated within the sources (for example by SQL-statements), or a customized servlet aggregates information from the documents retrieved. Following the aggregation the next step in processing federated searches is the gathering and combination of the ranked result sets from each data source. Every source delivers an ordered result list (often referred to as output stream), that can sometimes even be delivered successively. The ranked results have to be merged in order to get the most relevant result for the search. Therefore overall scores have to be aggregated using the scores from each source. Due to complex multi-feature queries a document perfectly matching one property may fail in all other properties and thus having an only medium or weak overall score. To get an overall order an aggregated score has to be assigned to every object taking into account the score values from every source. As was shown in Chapter 2, there are quite efficient ways to perform this task guaranteeing a correct overall result set without having to access all objects in each data source.

The way how results are integrated obviously is strongly user dependent. In complex queries the user might want to emphasize some properties, i.e. weighting the query, or use a special combining function. Such functions can be for instance the arithmetic mean to ensure a certain consistency of performance in all query properties for the top-scored objects or minimum /maximum to find the top performers in any query property. Thus not only the query properties themselves, but also user
Missing Scores in Quasi-Homogeneous Environments

Information portals have to deal with a variety of different information sources. Aggregating scores for objects in quasi-homogeneous environments can, however, be a major problem, if the score for any object in at least one of the atomic output streams is missing. A federated search in such environments would cause severe problems, since the request to any source for the score value of a certain object is a basic operation for calculating the object’s overall score. Consider for example a large digital image archive containing besides text sources and image annotations, two different collections of images on a certain subject, which may have photographs occurring in both collections, as well as some images occurring in only one of them.

These different collections may of course be indexed differently, for instance the first of them might be indexed for color-based and texture-based retrieval, whereas the second collection has been indexed for color-based retrieval only. The feature extraction for colors in this case (and generally for any shared feature) can be assumed to be the same or at least quite similar. Otherwise objects occurring in both collections may have two different scores for the same feature, which would have to be dealt with either in a preprocessing step, when querying the score value, or directly in the evaluation of the combining function. Although these problems occur quite often in the area of multisensor information fusion [MCGF99, XC99], they are not a real problem for the application here and we will apply the above assumption for our retrieval systems.

Also the problem of identifying objects over different sources to be two instances of the same object is not addressed here. It is assumed that when building the federated information system instances of the same object have been marked with the same unique OID taking some common attributes into account. Although we are assuming that instances of the same object are marked and have equal scores for the same features, there is still the problem of missing scores to solve. Consider a complex sample query on the image archive addressing the subject of our two collections from above. Whenever a user is looking for e.g. a picture meeting certain criteria and showing a
certain pattern, a federated search would correctly be started on our two collections as only they contain images on that subject. Objects meeting the properties have to be collected and a query on texture has to be posed. However, the texture-based scores are missing for all those images in collection 2, that are not included in collection 1, too. Of course, the quality of the overall retrieval result will suffer if there are too many unknown score values, but in general since we only look at the first few objects in each result set, the hope is that not too many scores will be missing.

In general there are three ways of calculating the overall aggregated score for these objects:

- Assigning a *fictitious score* for the missing value
- *Ignoring* the atomic query in the combining function for those objects not having a score
- Assigning *NULL-values* for missing scores (which would need a trivalent logic)

We will focus only on the first two alternatives, as the last one still leaves the question of getting an aggregated score. Such the choice has to be made between assigning fictitious scores or totally ignoring the missing feature. We will first have a closer look at assigning somehow estimated values for missing scores. There are mainly four approaches to estimate the score values:

- The *pessimistic approach* estimates the missing score as the worst score possible for the missing feature. Thus the overall scores for partially unknown objects are decreased and no difference is made between objects having a definitely low score and those having an unknown score. As only objects with a guaranteed high overall score will be returned, this case may be used in retrieval systems with emphasis on result sets showing high precision, though relevant objects might be left out of the result set.

- The *optimistic approach* estimates the missing score as the best score possible for the missing feature. The overall scores for partially unknown objects are increased and irrelevant objects may be included in the result set. However, at least no relevant object is left out. This case may thus be used in retrieval systems with emphasis on result sets showing high recall.

- The *balanced approach* will assume a medium score for the missing feature. A medium score can always be calculated as for example the average score of all
6.3 Missing Scores in Quasi-Homogeneous Environments

objects regarding the feature. Though this may be an applicable approach for combining functions like the arithmetical mean, this approach seems not to be too useful if combining functions as the minimum or maximum are involved.

- The **consistent approach** will assume the consistency of performance for any object, i.e. the estimated value for the unknown score is calculated based on the object’s scores for all the known features. A simple way to estimate an unknown score \( x \) would be \( x := F(x_1, \ldots, x_n) \) with the combining function \( F \) and \( x_1, \ldots, x_n \) being all the known scores of the objects. A far more complex, but also more adequate way would be the mapping of the object’s known performance on the distribution for the unknown feature. So the object’s degree of match or average rank can be used to determine the score of an object with the respective rank or degree of match in the missing feature’s result set. The value gained can then be used to estimate the missing score.

On the other hand, missing scores can simply be left out of the aggregation of scores. For virtually all sensible combining functions it can be shown that leaving missing scores out is just the same as assuming an object’s consistency of performance \( x := F(x_1, \ldots, x_n) \) as suggested above in our **consistent approach**. This theorem will be proven for weighted arithmetical means and can easily be generalized to other combining functions as average, minimum, maximum, geometrical means, etc.

**Theorem 8 (Ignoring missing scores is assuming consistency of performance)**

Without loss of generality let \( x_n \) be an unknown score in a set \( x_1, \ldots, x_n \) of scores for any database object. Given weights \( w_1, \ldots, w_n \) satisfying \( \forall i : w_i > 0 \) and a combining function \( F_{Q_1, \ldots, Q_n}(x_1, \ldots, x_n) := \frac{w_1}{w_1 + \ldots + w_n} x_1 + \ldots + \frac{w_n}{w_1 + \ldots + w_n} x_n \) ignoring a missing score \( x_n \) in \( F_{Q_1, \ldots, Q_n} \) can be done by setting \( x_n := F_{Q_1, \ldots, Q_{n-1}}(x_1, \ldots, x_{n-1}) \).

**Proof:** With \( x_n = F_{Q_1, \ldots, Q_{n-1}}(x_1, \ldots, x_{n-1}) \) we have to show that \( F_{Q_1, \ldots, Q_n}(x_1, \ldots, x_n) = F_{Q_1, \ldots, Q_{n-1}}(x_1, \ldots, x_{n-1}) \) and vice versa.

\[
x_n = \frac{w_1}{w_1 + \ldots + w_{n-1}} x_1 + \ldots + \frac{w_{n-1}}{w_1 + \ldots + w_{n-1}} x_{n-1}
\]

\[
\iff (w_1 + \ldots + w_{n-1}) w_n x_n = w_n (w_1 x_1 + \ldots + w_{n-1} x_{n-1})
\]

\[
\iff (w_1 + \ldots + w_{n-1}) \left( \frac{w_1}{w_1 + \ldots + w_{n-1}} x_1 + \ldots + \frac{w_{n-1}}{w_1 + \ldots + w_{n-1}} x_{n-1} \right) + w_n x_n =
\]

\[
= \left( \frac{w_1 + \ldots + w_{n-1}}{w_1 + \ldots + w_n} \right) (w_1 x_1 + \ldots + w_{n-1} x_{n-1})
\]

\[
\iff \frac{w_1}{w_1 + \ldots + w_n} x_1 + \ldots + \frac{w_{n-1}}{w_1 + \ldots + w_n} x_{n-1} = \frac{w_1}{w_1 + \ldots + w_{n-1}} x_1 + \ldots + \frac{w_{n-1}}{w_1 + \ldots + w_{n-1}} x_{n-1}
\]

\[
\iff F_{Q_1, \ldots, Q_n}(x_1, \ldots, x_n) = F_{Q_1, \ldots, Q_{n-1}}(x_1, \ldots, x_{n-1}) \quad \square
\]
What kind of approach to deal with missing scores is eventually chosen, strongly depends on the area of application for the retrieval system and the expectations and specific needs of its users. But the approaches show that the problem can almost always be solved in a sensible way and estimated scores can be provided for the missing ones. By the above considerations we may therefore assume for the following case study, that aggregated scores can always be calculated over quasi-heterogeneous information sources without any missing scores.

### 6.4 Case Study: Prognosis from Current Traffic Situations

For the case study we present a quasi-homogeneous environment that is quite characteristic for broader portal applications. We will, in the following, focus on deriving a prognosis for traffic situations. We therefore assume that we have a portal to a variety of data sources which contain all the data that is relevant to analyze traffic situations. We further assume that a federated search in all these sources is possible and that data sets in all of the sources are marked by an standardized location and time (e.g. GPS data and TIMESTAMP) to establish the quasi-homogeneous environment. The process of getting an adequate prognosis is divided into two steps:

- First we have to look through all the sources and find traffic situations that are in some aspect most similar to the current situation. Data sets for different aspects often will be retrieved from different sources and even publicly available sources from the Internet may be involved. Eventually we will get a set of best matching situations with the grade of match assigned.

- In the second step we will take a closer look on what happened in these past situations and try to find a pattern that will help us establishing a prognosis for the current situation. The hope of course is that similar causes will lead to similar developments, such that if past situations are ’similar enough’ in several aspects, their development will be a sufficient model to predict the current development.

To perform our first step we have to chose the data sources which are necessary to solve our problem. Since our choice strongly depends on the problem, its domain, our viewpoint and our idea of solving it, we first have to get an adequate set of data
6.4 Case Study: Prognosis from Current Traffic Situations

sources. In portal technology choosing sources is always considered an important step, because the portal has to provide a personalized view on data. We also have to choose a combining function and an adequate retrieval method for each source (e.g. visual retrieval for aerial photographs).

Like in the case of content-based retrieval also the similarity for the application in traffic prognoses is a n-dimensional problem. To get a good prognosis many aspects must be taken into account. The quality of the prognosis derived will strongly depend on a variety of features. For our case study we will provide only a few examples, that, however, do not claim to be exhaustive. Traffic situations may obviously be similar in the aspect of

- local traffic data (e.g. sensor data)
- season
- local events
- weather

For instance the current state at the same location may be of special interest, if there is e.g. road works or a blocked street. In this case the traffic situation of the last few days will strongly influence the prognosis. Current events like e.g. accidents will generally also strongly influence the prognosis, but their use depends on the update rate and timeliness of our information sources. The situation may also be similar in seasonal effects (e.g. increased volume of traffic during Christmas time). Thus the traffic situation on similar dates in the preceding years may be of major importance for the prognosis. Another helpful source is knowledge about various local events (e.g. trade fairs, sport events, etc.) that may cause traffic jams or even block some roads. A data source containing data about e.g. the average number of visitors for each event helps to model the expected change in terms of volume of traffic. For our example the last similarity will be weather information. Obviously the traffic will be strongly influenced by e.g. heavy rain or black ice. Thus the situation at one specific location may totally differ from the situation a week ago or even a year ago. Weather information is a typical example of publicly available information sources that can be used by different applications for a variety of purposes.

Having stated the notion of similarity, constructed the combining function and chosen the necessary sources, the next aim is to provide all the features of the current traffic situation that will be needed for the similarity search. For instance in our
previous example we would have to provide location, time, current events and the weather conditions. The complex query for most similar situations can then be split into several atomic queries and the ranked result sets delivered by each source can be merged using the Quick-Combine or Stream-Combine algorithm. For the merging we can use the standardized locations and timestamps to calculate aggregated scores with our customized combining function. Eventually we will get a ranked list containing those situations most similar with respect to most of the aspects.

With the first few elements from this list we can retrieve the developments at the respective time from a traffic archive. These developments have to be compared and can be the basis for a reliable prognosis of future traffic situations. The level of certainty or reliability of this prognosis can again be derived from the aggregated similarity scores. The more similar the past situation to our current situation is, the more probable an equivalent development will be. The quality of the prognoses also strongly depends on the area of application, since many areas will still be far too complex to model or are not yet understood well enough to derive reliable prognoses.

The area of application, however, is not restricted to traffic prognoses. Nearly the same considerations are useful for applications ranging from medical applications (persons with similar characteristics and similar symptoms may suffer from the same
6.4 Case Study: Prognosis from Current Traffic Situations

illness or need a similar therapy) to enviromental studies (similar kinds of pollution in similar ecosystems may have similar effects to those in the past). The combination of ranked result sets thus also can lead to high-quality information in the case of portal applications, if the portals data sources provide some common attributes.
6 Applications in Information Portals
7 Summary and Outlook

7.1 Summary

In the discourse of this work we addressed the problem of combining ranked result sets. These specific kinds of results are very common in the framework of modern multimedia retrieval. This typical new retrieval model uses, unlike traditional relational database systems, the similarity of database objects in certain aspects as basis for the retrieval. However, these ranked result sets consist of all the database objects with a grade of match (score value) attached. To get the overall best objects from a set of retrieval systems or datasources aggregated score values have to be calculated using any monotonic combining function. Calculating aggregated scores for all the retrieval results, i.e. for all the database objects, would require linear scans over the database. Thus the combination of ranked result sets is a severe performance bottleneck.

To improve the performance of modern retrieval systems we proposed two algorithms – called Quick-Combine and Stream-Combine (both covered by patent EP 00102651.7) – to combine multi-feature queries [GBK00, GBK01]. For practical applications it is of course essential to know which objects from different sources or retrieval systems belong together to allow the calculation of aggregated scores. We distinguish between three different cases:

1. Retrieval in homogeneous environments, where objects in different data sources have common keys

2. Retrieval in quasi-homogeneous environments, where objects in different data sources have some standardized attributes in common

3. Retrieval in heterogeneous environments, where objects have to be matched with each other using a complex function or comparison procedure
Summary and Outlook

In general the ranked retrieval model provides two different ways to get an object with its score: The sorted access relies on iterators that iterate the result set and always provide the next-ranked object. The random access provides the score for a specified object in any of the data sources. Since the last two retrieval scenarios show severe problems with access structures for random accesses [Coh98, WHRB99] our algorithm Stream-Combine was specially designed to rely on sorted accesses only. However, in general Stream-Combine will access far more objects in total than Quick-Combine. Thus the choice between both algorithms essentially depends on the type of environment and the costs of the comparison procedure. But whatever algorithm is chosen, it can be stated that 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.

To get empiric results on both algorithms’ performance we examined previous work in this area and compared our approaches to Fagin’s algorithm presented in [Fag96]. The Quick-Combine algorithm was proven to efficiently retrieve the $k$ most relevant database objects with guaranteed correctness for every monotonous combining function. Besides its control flow adapts flexibly to different score distributions. 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 of around 30. The empiric performance results also showed that the algorithm scales with both an increasing number of streams to combine and the database size. These remarkable benchmark results suggest that a real performance breakthrough for applications like content-based image retrieval in large image databases is in sight.

For the Stream-Combine algorithm we also proved the correctness of the result set and provided a worst case estimation for the performance. Stream-Combine is due to its control flow self-adapting to the retrieval scenario as well as different score distributions. Besides it can use every monotonic combining function to aggregate scores. Since Stream-Combine is designed as a middleware algorithm, the problem in providing a real world benchmark is to estimate the number of objects in each streams that have to be transferred to the middleware. Generally speaking, due to the
7.2 Outlook and Future Work

worst case estimation especially for skewed score distributions in the output streams, a major performance gain over today’s retrieval performance [WHRB99] can be expected.

As far as the applications for combining algorithms are concerned, we have pointed out several examples for applications in different environments. Originally the algorithms were developed for the content-based image retrieval in digital libraries [KEUB+98]. This application is an important domain, because the new SQL/MM standard implements combinations of several content-based features in so-called feature-lists. But combining algorithms are equally useful in a variety of applications. We have shown multi-classifier combination in homogeneous environments and portal technology for the quasi-homogeneous case [BGK00]. As the ranked retrieval model gets more and more important it seems certain that the number of useful application domains will even considerably increase in the near future.

7.2 Outlook and Future Work

Let us conclude with a short outlook. The work on combining ranked result sets is important in a variety of architectures. Combining algorithms may be implemented inside database kernels, in middleware architectures or even as a client application featuring several retrieval engines. As stated above, combining algorithms will gain speed together with the further improvement of the underlying retrieval systems and especially with the development of high-speed iterators and access structures. These operators are implemented inside each retrieval system’s kernel and can therefore not be influenced or tuned from middleware applications. However, the implementation of combining algorithms inside the kernel would be rather a difficult and very expensive task, though it might provide considerable speed-ups in terms of necessary retrieval time. Therefore the development of adequate middleware combining algorithms considering the underlying systems as black boxes will be an important task. First approaches of this kind have been sketched for the application of multi-classifier combination.

Future work will thus mainly focus on further improvements of the retrieval and the development of heuristics to estimate how many objects have to be retrieved when running the combining algorithm in a middleware tier. Especially the last task will provide us with more exact estimations of the performance of algorithm StreamCombine. It will also help us to decide which of the different retrieval strategies of
Summary and Outlook

If an estimation could be gained of how many random accesses in Quick-Combine can be performed that still guarantee a better performance than using Stream-Combine only, then even mixtures between both of the retrieval strategies seem possible. Our research states that the heuristics used for the indicator computation will also have a strong influence on the decision, whether a stream should be expanded any further by sorted access or whether certain objects’ scores should be gained by random accesses on that stream.

Another important task is the application of combining algorithms in new and challenging domains. Portal technology and applications for semi-structured data, especially together with the application of XML, provide a lot of open problems that could be addressed using sophisticated heuristics from the information retrieval domain, e.g. [KZSD99, Con99]. Due to their sophisticated control flows different heuristics and retrieval strategies can be easily adopted by both of the algorithms. Further interesting areas of application would be in time series [GK95] and content syndication [SML99].

A second area of improvement for today’s retrieval systems is the application in high-dimensional domains. Generally speaking, if the similarity between objects is described more accurately, the similarity can be guaranteed in more aspects leading to a high-dimensional feature space [BBKS00, RMF00, BEK98]. Our hope is that a combination of result sets can be an important step towards breaking the curse of dimensionality that always occurs in high-dimensional retrieval problems. It seems by far easier (especially if skewed score distributions are assumed) to combine a small number of objects, that have been retrieved using low dimensional feature spaces, than relying on high-dimensional access structures that only too often deteriorate with increasing number of dimensions [BBK98, FBF94]. High-dimensional access structures have even been proven to be often worse than linear scans, if only the number of dimensions is considerably high. Besides, the use of indicators in our combining algorithms prefers the most discriminating or high-weighted streams, i.e. dimensions, for expansion. This means that in high-dimensional cases some dimensions possibly might not be expanded at all. To state if combining algorithms are able to overcome this problem, however, will need deeper research as well as the development of sophisticated heuristics for an adequate retrieval performance.
## Acknowledgements

I am grateful to Prof. Ulrich Güntzer and Prof. Werner Kießling for their support in our joint research work. I would also like to acknowledge Matthias Wagner (especially for his very nice \LaTeX{} style), Achim Leubner and Ortrud Wenzel for proofreading and lots of helpful comments and suggestions.

The patent for the algorithms Quick-Combine and Stream-Combine was supported by the DFG-Ideenwerkstatt. The HERON-project is founded by the German Research Foundation DFG within the strategic research initiative "Distributed Processing and Exchange of Digital Documents". The database software applied in the HERON system is granted within IBM’s program "DB2 For Educational Purpose".
7 Summary and Outlook


Bibliography


Bibliography


Bibliography


Bibliography


