Towards Efficient Multi-Feature Queries in Heterogeneous Environments

U. Güntzer[1], W-T. Balke[2], W. Kießling[2]
[1]*University of Tübingen,* [2]*University of Augsburg, Germany*
guentzer @informatik.uni-tuebingen.de, {balke, kiessling}@informatik.uni-augsburg.de

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. We present a new algorithm Stream-Combine for processing multi-feature queries on heterogeneous data sources. Stream-Combine 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.

1. Introduction

Modern databases and information systems increasingly rely on a ranked query model. In this model for each query a grade of match is assigned to every database object. The query result is not a fixed set of objects, but rather a list of all database objects ordered by descending score values. These sorted lists are often referred to as *output streams*, since database objects are delivered one by one starting with the best objects. An output stream generated for only one feature is called atomic and any complex query can be answered by combining the output streams for all atomic parts of the query.

This ranked query model has proven its advantages especially in fields where similarity between objects is vital, e.g. document relevance regarding a set of keywords or visual similarity between a set of images. In these cases similarity does not only depend on a single feature, but rather is approximated by a set of different classifiers. Each classifier assigns different score values and thus ranks the database objects individually. The relevance of any object can then be determined by aggregating the individual (*atomic*) scores using any suitable monotonic combining function. The ranked query model is especially useful for handling multimedia documents and has already been used for retrieval facilities within the new SQL multimedia standard (SQL/MM [11]). However, getting only some overall best objects out of the ranked result sets without performing a complete database scan has remained a difficult problem. When it comes to retrieval in heterogeneous environments (e.g. classifiers from different information systems or retrieval across different data sources) the problem is still largely unsolved.

Recent research in multiple classifier combination [7] focuses mainly on the development of meaningful classifiers or tries to determine most effective combinations of these classifiers by assigning weights and optimizing the combining function. In applications like enterprise information portals and digital libraries today's problems, however, have shifted from the mere effectivity of retrieval towards efficiency issues. Considering for example combinations of multiple classifiers for speech recognition there are immense numbers of candidates for recognized sentences even for small grammars [8]. Thus the aim must not only be to find more adequate classifiers, but also to improve the efficiency by finding the overall best candidates without looking at the entire set of possible sentences.

2. Related Work

General middleware like IBM's GARLIC [1], visual retrieval systems like HERON [2] and all kinds of Internet sources or enterprise information portals [3, 6] 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. Consider e.g. the architecture of the HERON system shown in Figure 1. It offers a middleware solution where a query engine is used for splitting up complex queries and posing them to the underlying database system that is extended by advanced content-based retrieval facilities. The Combining Engine collects all the incoming output streams using the Quick-Combine algorithm [4] to find the top-scored objects most quickly. 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 [5].

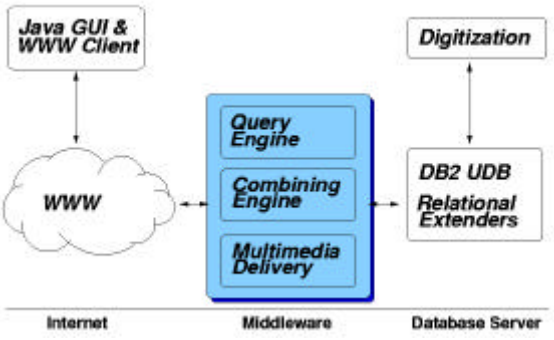

Figure 1: Architecture of the HERON system

Previous approaches in the field of multi-feature query combination can generally be divided in statistical approaches [9] and those with guaranteed correctness [4, 10]. Whereas statistical approaches assume certain score distributions in the output streams and estimate how far the streams have to be fetched to calculate the best objects with arbitrarily high probability, the other type of algorithms can guarantee a correct result set and even benefit from the real score distribution in each stream. As these are favorable qualities, we will in the remaining of this paper only refer to algorithms delivering correct result sets.

All of the above algorithms rely on two main operations:

- Sorted access
- Random access

Sorted access means to access objects by getting the next-ranked object from any of the output streams (or a larger, suitable number of objects ordered by their ranks, if bulk access is feasible) . In contrast, a *random access* is performed, if the specific score value for an already seen object is retrieved from any classifier. However, in heterogeneous environments doing random accesses may be very expensive or sometimes even impossible [5, 6]. The algorithms mentioned above may differ in the exact number of random accesses, but all of them need a considerable number of random accesses before termination. In the following we will present a new algorithm retrieving the top *k* objects by combining *n* different atomic output streams *without* needing any random access. Besides, our algorithm can guarantee a correct result set and will output objects *successively* as soon as they are found.

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 [6]. 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. 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.

Algorithm Stream-Combine (Basic Version)

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

1. **Initialization:** Get some first objects 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 and 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 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 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 a new aggregated score has to be calculated if all exact local scores are known or estimated using the upper bounds from step 3.
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 1: *If an object o_{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_{top}. Because o_{top}'s aggregated score is supposed to be larger than the upper bounds for aggregated scores 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_{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 or equal than the lowest scores seen in each stream so far and due to the monotonic combining function the aggregated score of o_{top} is larger or equal than the score of o, which is contrary to our assumption.

A short example will help us to understand how Stream-Combine works and will lead to heuristics for further improvements. To answer a complex query for instance 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.

q_1 : query on keyword						
Rank	1	2	3	4	5	...
Score	0.98	0.93	0.71	0.71	0.70	...
Object	O4	O5	O6	O3	O7	...

q_2 : query on visual similarity						
Rank	1	2	3	4	5	...
Score	0.96	0.88	0.85	0.84	0.83	...
Object	O1	O2	O3	O4	O5	...

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*):

Object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	0.98	yes	0.96	no	0.97
O1	0.98	no	0.96	yes	0.97

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:

Object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	0.98	yes	0.85	no	0.915
O1	0.71	no	0.96	yes	0.835
O5	0.93	yes	0.85	no	0.89
O2	0.71	no	0.88	yes	0.795
O6	0.71	yes	0.85	no	0.78
O3	0.71	no	0.85	yes	0.78

The next expansion of stream 1 reveals the missing score for the already known object O3:

Object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	0.98	yes	0.85	no	0.915
O1	0.71	no	0.96	yes	0.835
O5	0.93	yes	0.85	no	0.89
O2	0.71	no	0.88	yes	0.795
O6	0.71	yes	0.85	no	0.78
O3	**0.71**	**yes**	**0.85**	**yes**	**0.78**

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:

Object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	**0.98**	**yes**	**0.84**	**yes**	**0.91**
O1	0.71	no	0.96	yes	0.835
O5	0.93	yes	0.84	no	0.885
O2	0.71	no	0.88	yes	0.795
O6	0.71	yes	0.84	no	0.775
O3	**0.71**	**yes**	**0.85**	**yes**	**0.78**

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. 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.

3.1. 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 earlier start working 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 1 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 example from above we have already calculated the final score of object O3 and found it to be amongst the five best objects. Thus if $k \geq 5$ had been chosen, we could already have output O3.

After the first object has been delivered, we have to find the next known object amongst the top (*k*-1) 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.

3.2. 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 $\partial F/\partial x_i$. 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 $s_i(o)$ 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:

$$\Delta_i = \#M_i \cdot \partial F/\partial x_i \cdot (s_i(r_i(z_i\text{-}p)) - s_i(r_i(z_i))) \quad (1)$$

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 1 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 needs to be updated. In our example from above after object O3 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.

4. Algorithm Stream-Combine (Full Version)

Given n atomic output streams q_1, ..., 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 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 and 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 ($1 \le m \le n$, $m \ne i$). 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 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 s 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 $\#M_i \ne 0$. 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 example from above. 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 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 $\partial F/\partial x_1 = \partial F/\partial x_2 = 0.5$ for the arithmetical means.

object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	0.98	yes	0.88	no	0.93
O1	0.93	no	0.96	yes	0.945
O5	0.93	yes	0.88	no	0.905
O2	0.93	no	0.88	yes	0.905

The score differences are: $s_1(r_1(2))-s_1(r_1(1)) = s_1(O4)-s_1(O5) = 0.98-0.93 = 0.05$ and $s_2(r_2(2))-s_2(r_2(1)) = s_2(O1)-s_2(O2) = 0.96-0.88 = 0.08$

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. $\#M_1 = \#M_2 = 1$. The indicators now can be initialized:

$\Delta_1 = \#M_1 \cdot \partial F/\partial x_1 \cdot (s_1(r_1(1)) - s_1(r_1(2)) = 0.025$
$\Delta_2 = \#M_2 \cdot \partial F/\partial x_2 \cdot (s_2(r_2(1)) - s_2(r_2(2)) = 0.04$

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

object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	0.98	yes	0.85	no	0.915
O1	0.93	no	0.96	yes	0.945
O5	0.93	yes	0.85	no	0.89
O2	0.93	no	0.88	yes	0.905
O3	0.93	no	0.85	yes	0.89

Now we have to calculate a new indicator for the expanded stream with $s_2(r_2(2)) - s_2(r_2(3)) = s_2(O2) - s_2(O3) = 0.88 - 0.85 = 0.03$. 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:

object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	0.98	yes	0.85	no	0.915
O1	0.71	no	0.96	yes	0.835
O5	0.93	yes	0.85	no	0.89
O2	0.71	no	0.88	yes	0.795
O3	0.71	no	0.85	yes	0.78
O6	0.71	yes	0.85	no	0.78

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:

object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O4	**0.98**	**yes**	**0.84**	**yes**	**0.91**
O1	0.71	no	0.96	yes	0.835
O5	0.93	yes	0.84	no	0.885
O2	0.71	no	0.88	yes	0.795
O3	0.71	no	0.85	yes	0.78
O6	0.71	yes	0.84	no	0.775

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:

object	s_1	seen in q_1	s_2	seen in q_2	upp. bound
O1	0.71	no	0.96	yes	0.835
O5	**0.93**	**yes**	**0.83**	**yes**	**0.88**
O2	0.71	no	0.88	yes	0.795
O3	0.71	no	0.85	yes	0.78
O6	0.71	yes	0.83	no	0.77

The final aggregated score of O5 can be calculated and is larger than all other scores. Thus O5 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 e.g. not have to calculate the irrelevant aggregated score of object O3 unlike the previous example.

5. Summary and Outlook

In this paper we presented a new algorithm, called Stream-Combine, that allows retrieving the *k* top-scored objects from a set of different ranked result lists without needing any random accesses. Since modern database and information systems tend to use a ranked query model, the efficient retrieval of only the top k overall best results is of course always vital. But especially in the field of heterogeneous datasources and retrieval systems current approaches will fail due to their excessive use of expensive random accesses. Existing algorithms are designed for homogeneous environments and have been shown to be even worse than the linear scan over all database objects. The innovative approach of Stream-Combine promises a considerable performance gain for heterogeneous environments by entirely turning down the need of any random accesses. Moreover, its indicator-based control flow provides useful heuristics to pick always those streams for expansion that promise the best improvement. Thus an early termination of the algorithm is forced. The set of best objects can also be retrieved successively such that first results can be delivered while the algorithm is still running.

Our future research will focus on studies on applications of Stream-Combine. The further development of high-speed iterators for sorted access in heterogeneous environments is also essential. Besides, real world benchmarks have to be obtained to get an better feeling for the complexity of Stream-Combine in practical applications.

6. References

[1] Cody, Haas, Niblack, et al. Querying Multimedia Data from Multiple Repositories by Content: The Garlic Project. In: Proc. of the Working Conference on Visual Database Systems (VDB-3), pp 124-131, Lausanne, Switzerland. 1995

[2] Kießling, Erber-Urch, Balke, Birke, Wagner. The HERON Project – Multimedia Database Support for History and Human Sciences. In: 28. Annual Conference of the German Computer Society (GI): INFORMATIK98, LNCS, pp. 309-318, Magdeburg, Germany, September 1998.

[3] Gravano, Garcia-Molina. Merging Ranks from Heterogeneous Internet Sources. In: Proc. of the Intern. Conf. on Very Large Databases VLDB 1997, pp. 196-205, Athens, Greece, 1997

[4] Güntzer, Balke, Kießling. Optimizing Multi-Feature Queries for Image Databases. In: Proc. of the Intern. Conf. on Very Large Databases VLDB 2000, pp. 261-281, Cairo, Egypt, 2000

[5] Wimmers, Haas, Tork-Roth, Brändli. Using Fagin's Algorithm for Merging Ranked Results in Multimedia Middleware. In: Proc of the Intern. Conf. on Cooperating Information Systems COOPIS'99, pp. 267-278, Edinburgh, Great Britain, 1999

[6] Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In: Proc. of the Intern. Conf. on Management of Data SIGMOD‘98, pp. 201-212, Seattle, USA, 1998

[7] Multiple Classifier Systems, LNCS 1857 (J.Kittler and F.Roli, Eds.), Springer-Verlag, 2000

[8] Yu, Jiang, Bunke. Combining Acoustic and Visual Classifiers for the Recognition of Spoken Sentences. In: Proc. of the Intern. Conf. on Pattern Recognition ICPR 2000, pp 491-498, Barcelona, Spain. 2000

[9] Chaudhuri, Gravano. Optimizing Queries over Multimedia Repositories. In: 16th ACM Symposium on Priciples of Database Systems, pp. 91-102. ACM 1997

[10] Fagin. Combining Fuzzy Information from Multiple Systems. In: 15th ACM Symposium on Priciples of Database Systems, pp. 216-226. ACM 1996

[11] ISO/IEC FCD 13249-5:1999 SQL/MM SAF-005. Information Technology - Database Languages - SQL Multimedia and Application Packages - Part 5: Still Image. 1999