Combining Metric Features in Large Collections

Michal Batko
Masaryk University
Faculty of Informatics
Brno, Czech Republic
batko@fi.muni.cz

Petra Kohoutkova
Masaryk University
Faculty of Informatics
Brno, Czech Republic
pkoh@ics.muni.cz

Pavel Zezula
Masaryk University
Faculty of Informatics
Brno, Czech Republic
zezula@fi.muni.cz

Abstract

Current information systems are required to process complex digital objects, which are typically characterized by multiple descriptors. Since the values of many descriptors belong to non-sortable domains, they are effectively comparable only by a sort of similarity. Moreover, the scalability is very important in the current digital-explosion age. Therefore, we propose a distributed extension of the well-known threshold algorithm for peer-to-peer paradigm. The technique allows to answer similarity queries that combine multiple similarity measures and due to its peer-to-peer nature it is highly scalable. We also explore possibilities of approximate evaluation strategies, where some relevant results can be lost in favor of increasing the efficiency by order of magnitude. To reveal the strengths and weaknesses of our approach we have experimented with a 1.6 million image database from Flicker comparing the content of the images by five similarity measures from the MPEG-7 standard. To the best of our knowledge, the experience with such a huge real-life dataset is quite unique.

1 Introduction

It is becoming common that practically any information is available in a digital form. Unfortunately, the exponential growth of the amount of digital data makes the searching for relevant pieces of information more and more difficult and the scalability of search engines is crucial.

The problem is further complicated by the complexity of digital data types – such as images, audio, time series, biomedical data, etc. – for which the traditional attribute-like search cannot be used. Instead, some characteristic *descriptor* is extracted from a complex object, e.g. a color histogram is computed for an image, and this simplified information is then used for searching. Depending on the objects' domain, several descriptors can be extracted to cover different aspects of the objects – for example colors, shapes and textures of an image. The descriptors are typically high-dimensional vectors or complex data structures for which nothing beyond pairwise distances can be measured, thus a distance-based index structure is required in order to process the data. The metric space model is a suitable abstraction that allows efficient indexing while still being applicable to wide variety of data types.

The Multi-Feature Indexing Network (MUFIN) provides an effective and efficient tool for searching in digital data collections which grow at exponential speed. The MUFIN is capable of executing similarity queries for any metric data and it allows definition of multiple search indexes (overlays) for different descriptors. In order to cope with large volumes of data and guarantee required performance constraints, a peer-to-peer (P2P) architecture and parallel processing is exploited in MUFIN.

To be able to execute also combined search where a query is composed of more than one descriptor and its result is formed by objects that best match the query in several aspects, we propose a distributed variant of the threshold algorithm (TA) for MUFIN. It allows a user to specify an arbitrary aggregation function for combining the individual descriptor distances into an overall distance so that the best matching results are closest to the query. The function can be different for each query thus different results can be obtained even if the query descriptors remain the same. To reveal strengths and weaknesses of running the TA in distributed environment, we conducted an extensive performance evaluation with a six-overlay MUFIN indexing 1.6 million descriptors in each overlay. The descriptors were extracted from images, thus the results can be easily visualized.

The paper is organized as follows. We describe the architecture of MUFIN in Section 2. Then, the distributed threshold algorithm is presented in Section 4. Section 5 reports on the different experiments we have conducted with the network and our experience is concluded in Section 6.

1.1 Related Work

The problem of executing complex queries, sometimes referred to as top-k queries, was studied by several researchers. The main attention was drawn to multimedia data [7, 17] and the domain of text documents [2, 14, 20]. The most successful approaches are based on the family of threshold algorithms [9, 11, 17]. Generally, these techniques are designed to run in a centralized database systems, but there are also extensions for distributed architectures such as peer-to-peer networks. Several techniques concentrate on structured data (e.g. XML documents, relational tables, etc.) and provide extensions for high-level query languages (such as SQL) [1, 13]. Unfortunately, such queries may produce huge numbers of sub-results in a large-scale peer-to-peer system and the process of merging the sub-results to get the final ranked result can be very expensive. Also, the impact of transferring the results using the network layer is becoming a major issue for greater number of peers. These implications are more extensively studied in [19], where a system for efficient combining ranked lists of XML documents is presented. Authors of [15] have proposed a framework called KLEE for processing top-k queries in distributed text-data repositories. The evaluation of ranked lists for a specific attribute (text term) is distributed to a network of peers and a probabilistic approximations are used on the true top-k result.

An approach from a different point of view is the M^3-tree structure [6]. It is a centralized index, where all features of a complex object are stored using a lower-bounding metric function in the M-tree [21]. Then, similarity query evaluation strategies are modified, so that they can use the lower-bounding property to get results for arbitrary complex queries.

However, all the aforementioned systems are based on attribute (term-based) values and cannot be used in generic metric space. The only exception is the M^3-tree, but its evaluation algorithms are quite expensive and the scalability quite limited. The MUFIN allows combination of any metric index structure with emphasis on the large-scale peer-to-peer systems and our distributed threshold algorithm aims at fast combined-query retrieval with tunable approximation.

2 Architecture

In this section, we outline the architecture of MUFIN. First, we explain paradigms used for data comparison and organization. Next, we briefly characterize the most important building blocks of the proposed architecture, i.e. the metric searching overlays. Finally, we show how the individual parts can synergically cooperate to achieve required objectives.

2.1 Fundamentals

MUFIN can deal with data compared on the basis of similarity, considering the exact matching as a special case. It accepts the metric vision of similarity [21] which measures closeness of objects as distance. In particular, the mathematical *metric space* is a pair (D, d), where D refers to the *domain* of objects and d is the *distance function* which is able to compute the distance between any pair of objects from D. It is assumed that the smaller the distance, the closer or more *similar* the objects are. For any three distinct objects $x, y, z \in D$, the distance must satisfy the following properties of *reflexivity*, $d(x, x) = 0$, *strict positiveness*, $d(x, y) > 0$, *symmetry*, $d(x, y) = d(y, x)$, and *triangle inequality*, $d(x, y) \leq d(x, z) + d(z, y)$. Such definition allows us to specify a number of similarity queries, including the similarity range and nearest neighbor queries.

It is quite obvious that the client/server technology currently widely used simply reaches its capacity and efficiency limits especially in the field of multimedia data. This problem can be overcome by employing several computers (formerly servers) connected via a network to cooperate while evaluating queries. However, data structures used for the server-based (centralized) solutions usually cannot be used directly in the networked environment – there are several issues (e.g. communication, navigation, reliability, etc.) that must be addressed.

A suitable implementation paradigm for such structures seems to be the P2P data network. The *peers* (computers participating in the network) offer the same functionality and the system follows the shared distributed logic that facilitates an effective intra-system navigation. In general, every peer of such system must provide its storage and computational resources, must be able to contact any other peer directly (provided its network identification is known), and must maintain an internal structure that ensures correct routing among the peers. For maximal scalability, there are three fundamental requirements: data expands to new peers gracefully; there is no master site to be accessed when searching for objects; and the data maintenance primitives never require immediate propagation of updates to multiple peers.

2.2 Distributed Metric-search Structures

In general, similarity searching is computationally demanding because the quadratic or even cubic time complexities of distance functions are not exceptional. Though there are many techniques proposed in literature [8, 12, 4], only the distributed versions can successfully deal with the scalability problem. For a metric space, there are several techniques [4, 10, 18, 5] able to deliver answers to similarity queries in P2P environment. Basically, they all follow the schema depicted by Figure 1. Every peer holds a partition

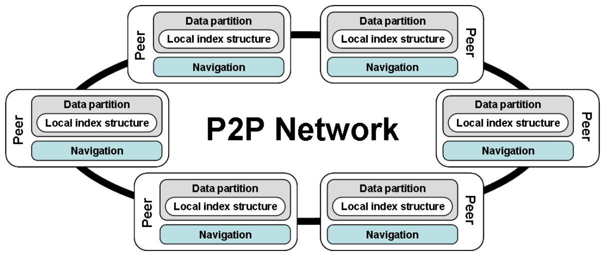

Figure 1. A typical schema of a P2P network

of the indexed data collection in its storage area. This data can be stored either as a simple list and a sequential scan applied during searching, or a sophisticated indexing technique can be employed. Besides, navigation knowledge is to be stored at every peer and it takes control of the mechanism to forward the query between peers. Peers communicate via messages which are delivered by the underlying computer network.

A user can issue a similarity query at an arbitrary peer and the steps depicted in Figure 2 are performed to answer the query. First, the peer consults its navigation knowledge to get a list of peers responsible for data partitions that can contain qualifying objects and forward the query to them. Since the P2P network can change in time, the navigation can be imprecise, so the query can be forwarded several times until it reaches the respective peers (solid arrows). At every peer with a promising partition (e.g., the query intersects the partition), a local search procedure is executed giving all the objects satisfying the query constraint (peers with star mark). Finally, all the contacted peers return their partial results to the originating peer (dotted arrows), where the final answer is merged and passed back to the user. If we can miss some qualifying objects (i.e.

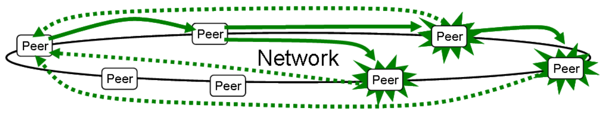

Figure 2. A typical query processing in P2P

sacrifice the precision), we can gain one or two orders of magnitude [22] faster evaluation – the *approximate similarity* search is used. The idea behind is quite simple. When forwarding the query to other peers, we do not send the query to all qualifying peers, but rather only to few most promising ones. Also, the local index structure can apply any form of local approximation strategy to further improve the speed. Of course, the result returned to the user can be incomplete.

2.3 Multi-layer System

As anticipated earlier, some applications require to solve aggregate similarity queries over several metric spaces. By combining multiple single-metric indexes, queries with combinations of more metrics can be answered. Owing to the requirement of retrieving objects relevant from several aspects' points of view, some kind of aggregation of partial results is necessary.

Assume that a complex data object is simplified by extraction techniques into a set of *descriptors*. Such a set together with an identifier of the original object form a *metaobject*. To compare similarity within respective descriptors, a metric function is specified. For instance, we would like to index images from which we can extract color and shape descriptions. A metaobject for a given image is composed of the two descriptors – one for the color and the other for the shape – plus the link to the original image.

Naturally, to express the similarity of two metaobjects, we must combine the similarities of their respective descriptors. For this task, a user-supplied *aggregation function* is used. The aggregation function is required to be monotonous. The result is not necessarily a metric (consider a simple aggregation function $f(x) = x + 1$, which is monotonous but the reflexivity property does not hold: $f(d(o, o)) = f(0) = 1$), but it expresses the similarity in the same way as metric functions – the higher the value of distance the more dissimilar the objects are. Following our example with colors and shapes, we supply a weighted sum as the aggregation function, i.e., the weighted sum of the color and shape descriptor distances. This means that we perceive two objects similar if they are similar in both color and shape and weights are used to emphasize either color over shape or the other way round.

To evaluate similarity queries efficiently, we build a P2P index for each of the descriptors. So, every single descriptor of a particular metaobject is stored by the respective index along with an identifier of the metaobject. Moreover, a special *zero overlay* is defined where complete metaobjects are stored. The zero overlay allows efficient retrieval of metaobjects using their identifiers as a key – a classical P2P distributed hash table can be used, because we only need the "get-by-id" operation in this overlay. In principle, these overlays are allowed to share the same infrastructure of physical peers.

Figure 3 depicts a system with three overlays. The first is built for color descriptors, the second indexes shapes, and the third represents the zero overlay. A metaobject assignment to these overlays is also illustrated in the figure. Observe that each overlay consists of multiple nodes and their specifics are left up to a particular distributed index structure used in the overlay. These nodes are maintained by physical peers (illustrated by the dotted arrows). Each

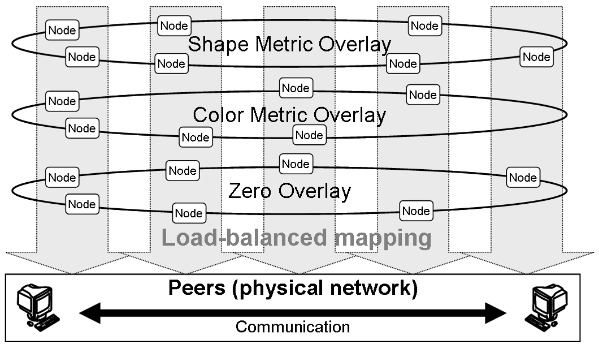

Figure 3. Multi-metric overlay setting

peer usually manages one node from every overlay. Such a mapping is completely transparent for overlay index structures and in general, it is automatically done by the load-balancing mechanism.

2.4 The MUFIN System

Since MUFIN is a general purpose software product, it can be applied to similarity search problems of a variety of applications. In order to organize a specific data collection (e.g., images, music, scientific experiments), a MUFFIN instance needs the following parameters to be specified for each of the data collection descriptors: (1) the *metric space* – the domain of the particular descriptor and its distance function; (2) *index structure* – the actual implementation of a single-metric P2P system used for the overlay; (3) *local storage index* – a wide variety of local metric index structures can be used to speed-up the processing within nodes of the overlay. In addition to, a distributed hash table for the *zero overlay* must be built.

3 Standard Threshold Algorithm

The *threshold algorithm* [9] (TA) was proposed for obtaining top-k results of several ranked lists. Even though it was originally proposed for the scores (rank), it can be inverted to suit the distance measures.

Let $Top(k, q, f)$ be the top-k query for the query object q and aggregate function f that we want to solve on a database with m descriptors $s_1, \cdots, s_m$. Note that each descriptor forms a metric space with its own metric function $d_1, \cdots, d_m$. Then, the *sorted access* of descriptor s_i is a list S_i of all objects and their distances to the query object using only the descriptor s_i. The list is sorted by distances, the lower the distance is the sooner the object appears in the list. On the other hand, *random access* can retrieve the distance between the query and a given object o for all the descriptors and thus compute the overall distance of this object as $d(q, o) = f(d_1(q, o), \cdots, d_m(q, o))$. Then, the algorithm works as follows:

1. Do sorted access in each of the m descriptors to get the sorted lists.
2. In every iteration get the next object from each sorted list $o_1 \in S_1, \cdots, o_m \in S_m$ having the respective descriptor's distances $\delta_1 = d_1(q, o_1), \cdots, \delta_m = d_m(q, o_m)$.
3. Use the random access to the other lists to compute the overall distances of objects $o_1, \cdots, o_m$. Update the list of the resulting k objects with the lowest overall distances. Let d^{max} be the distance of the k^{th} object, i.e. the maximal distance of the result.
4. Compute the threshold value $t = f(\delta_1, \ldots, \delta_m)$. Do next iteration unless the result list has k objects and $d^{max} \leq t$.

The correctness of the stop condition in the fourth step is proved in the original paper [9] and the modification for metric distances is straightforward. Basically, the threshold t in each iteration of the algorithm can only increase while the maximal distance d^{max} only decreases after the result list is filled with k objects.

4 Distributed Threshold Algorithm for MUFIN

Running a standard TA in a distributed environment would be very expensive, because a single object retrieval is very inefficient. The batch approach is more suitable and it works as follows (see Figure 4 for schematic overview). The issuing peer breaks the query metaobject into its descriptors and executes a nearest neighbor query for every descriptor in the respective similarity-search overlay. They are evaluated in parallel and a sorted list of the top-most similar objects is returned for each descriptor. The objects are then used to query the zero overlay to get distances for missing descriptors. Next, an aggregation function is used to compute the objects' overall similarity. If there are not enough objects with their overall similarity under a certain threshold value, the descriptor overlays are requested to provide additional batch of objects until this condition is met.

However, to get under the threshold can take a lot of time for a huge data collections. For example, a top-50 query in a dataset of 1.6 million images takes more than one minute to evaluate even for a batch size of 1,000 objects. Moreover, the interpretation of similarity itself is highly individual so even the optimal results of the search may not satisfy user needs. Therefore, it is more reasonable to give good-enough

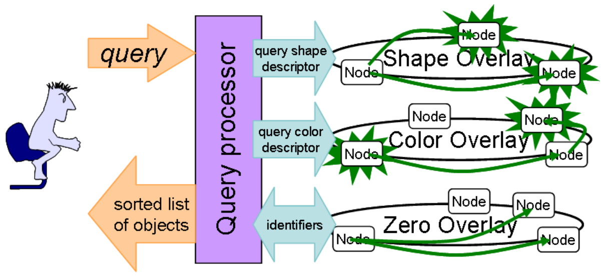

Figure 4. Evaluation of a complex query

results quickly even if they are not precise. Thus, we alter the stop condition and we end the processing prematurely after ϵ iterations even if the threshold condition is not satisfied. The value ϵ allows us to tune the ratio of the response time and the quality of the result.

Since the parameter ϵ is specific for each query, the system can automatically adjust the value according to user preferences or the actual system load. To help the system tune the ϵ more precisely, we can compute actual quality estimations during the iterations of the TA. Since the actual threshold value t and the maximal result-list distance d^{max} are updated in every iteration of TA, we can see them as functions $t(i)$ and $d^{max}(i)$ of the TA iteration i. If we know the final maximal distance $d^{max}(final)$ of the precise result, we can express the quality of the result after i iterations as a ratio $d^{max}(i)/d^{max}(final)$. The best quality is equal to 1 (precise result) while the higher values represent worse quality. However, the final distance is unknown during the evaluation and we can only use the threshold $t(i)$ as its lower bound (due to the TA stop condition). The quality is therefore upper-bounded by $d^{max}(i)/t(i)$. Using the first ϵ values of $t(i)$ and $d^{max}(i)$, we can improve the estimation of the quality by extrapolating the behavior of the $t(i)$ and $d^{max}(i)$ functions. Then, their intersection can be computed, and the function value at the intersection is an estimation of the $d^{max}(final)$ and can be used to compute the estimated quality at iteration ϵ.

5 Experiments

For the experiments, we have used an instance of MUFIN system loaded with 1.6 million images gathered from the Flickr[1] system. Generally, the images are outdoor and indoor taken photos, but there are also few images of e-shops products (e.g. mobile phones, PDAs, CD players, etc.), cartoon characters or hand drawings and paintings. Images are described by five descriptors defined in the MPEG-7 standard [16] that give evidence about the visual content of the images. The MPEG-7 standard defines also the metric distance measures for each descriptor. The following table summarizes the used descriptors and the metric functions.

MPEG-7 Feature	Distance	Weight
Scalable Color	L_1 metric	2
Color Structure	L_1 metric	3
Color Layout	special	2
Edge Histogram	special	4
Homogeneous Texture	special	0.5

In order to be able to compare results of our distributed threshold algorithm, we have fixed the aggregation function to be the weighted sum of all the five descriptors – the respective weights form the last column of the table.

There were 77 peers in our MUFIN instance, each holding up to five logical nodes of any of the six overlays. These peers were run on a physical infrastructure with 16 CPUs (AMD Opteron 2.6MHz) and 64GB RAM in total. For the nodes indexing metric overlays we have used the GHT* [5] structure. Each node has used a local M-Tree [21] to actually store the descriptors. For the zero-overlay, we have used the Skip-Graphs [3] distributed hash-table.

5.1 Standard Threshold Algorithm

Before we could start working on approximations, we had to understand the standard TA well. Therefore the first experiments were focused on describing the evolution of the result set and the threshold value during the execution of the algorithm. In these experiments, we logged the result sets (i.e. objects in the result set and their distances from query object) and threshold values computed in individual iterations of TA.

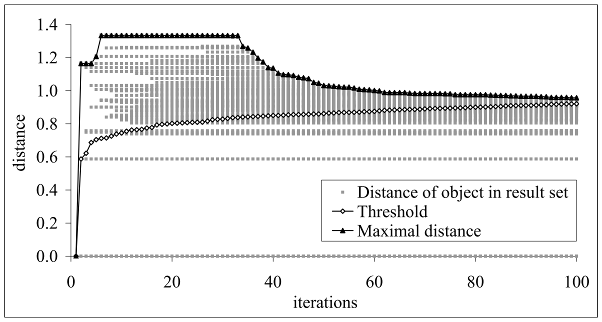

Figure 5. Visualization of the standard TA

The evaluation of a top-50 query is visualized in the graph in Figure 5 for iterations 1 to 100. All five descriptors are combined using the aforementioned weighted-sum aggregation. Note, that in each iteration only one step of the standard TA is computed. Gray squares represent distances of objects in the result set in that iterations. The upper curve

[1]http://www.flickr.com/

represents the maximal distance $d^{max}(i)$ in iteration i, the lower curve shows the respective threshold value $t(i)$. Observe, that the $d^{max}(i)$ curve first increases until the initial 50 unique objects are found. Since we have five sorted lists, it can happen earlier than in the 50^{th} iteration – iteration #33 in our case. After that, the curve can only decrease, because the result can only be updated with closer objects. On the other hand, the threshold can only grow. The intersection of the two curves marks the stop condition of the standard TA. The distance between the curves after each iteration corresponds to the quality of current result set. Experiments confirmed that the quality increases very quickly in the first few iterations while the rate slows down later. This observation forms the basis of our approximation.

5.2 Distributed Threshold Algorithm for MUFIN

In this section, we report on our experience with running the distributed TA. We have executed top-50 queries combining all five descriptors using the weighted sum aggregation function. To compensate for local irregularities every experiment was repeated for 50 different randomly chosen query objects and the average results are reported. We have also evaluated each query using a sequential scan in order to establish a baseline – precise answer to a top-k query. This is then used for computing recalls and distance errors of the approximate search.

In the first experiment, we have measured the performance of our complex query evaluation with respect to the number of sorted accesses used in one batch. The approximation parameter ϵ is set to the size of the batch, thus always only one batch is executed. Two measures were observed: the response times (Figure 6) and the number of distance computations (Figure 7). To establish a baseline, the following table summarizes both the measures for the sequential scan, parallel sequential scan run on 16 machines and a GHT* index using the weighted sum aggregation as a single metric function. Note that the evaluation of one object needs in fact five distance computations – one for each MPEG-7 descriptor.

Approach	Time	Distances
Sequential scan	2 mins	8,000,000
Parallel seq. scan	7 secs	500,000
Single-metric GHT*	1.9 secs	244,770

Response time describes the average overall time of query processing when ϵ sorted accesses are used. The time includes not only the sorted accesses run in parallel, but also the time needed for random accesses and computations. We can clearly see that the time is linearly proportional to the batch size. This is an expected behavior, since the number of objects that must be processed increases linearly with the batch size. However, even the biggest batch size was evaluated faster than both the sequential scans. On the other hand, a distributed index structure with fixed metric was faster except for very small batch sizes, but it lacks the flexibility of changing the aggregation function.

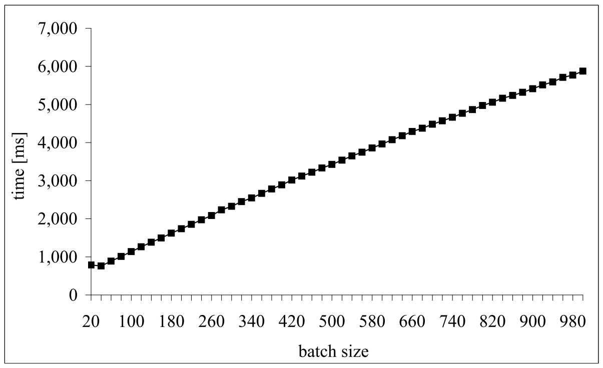

Figure 6. Response times using MUFIN

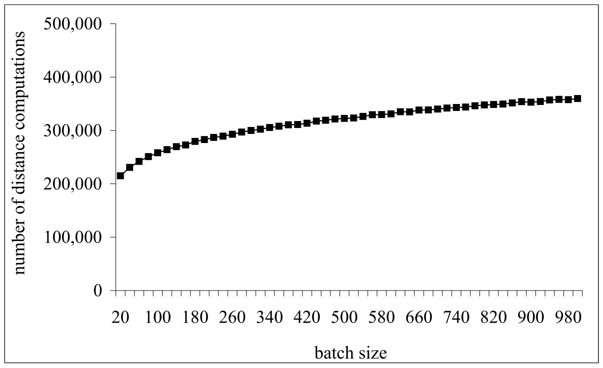

Figure 7. Distance computations in MUFIN

We have also measured the *number of distance computations* necessary for all the sorted and random accesses. This measure is proportional to the CPU load, since the distance computations are by far the most expensive operation. We can see that small batch sizes require rather high number of distance computations and as the batch size increases, the costs are not increasing that fast. It is usually expensive to find the first few most similar objects, because many partitions must be visited even thought they do not contain relevant objects. Increasing the number of searched objects usually does not require visiting many additional partitions, which is a well-known fact. However, the time saved on distance computations when the batch size increases is spent on maintaining the bigger lists of objects (e.g. sorting) and by the communication between the peers.

In the second group of experiments, we have focused on the quality of the approximation. *Recall* (see Figure 8) shows the percentage of objects of precise result that are present in the result given by the approximation. The true recall can only be computed once we know the exact result. If this value is unknown, which is the case while the approximation is evaluating, we can only estimate the recall. The estimation based on the last computed threshold value is called *Upper bound* in the graph. Using the extrapolated value (using linear extrapolation of the threshold and maximal distance curves), we can further refine a better value denoted as *Recall estimation* (see Section 4 for details). The recall of 20% for small values of ϵ is very low and such approximation would not be useful; however, recall of 70% and better, which is achieved for ϵ greater than 250, is quite sufficient, especially when the differences between distances of objects in exact and approximate result are small.

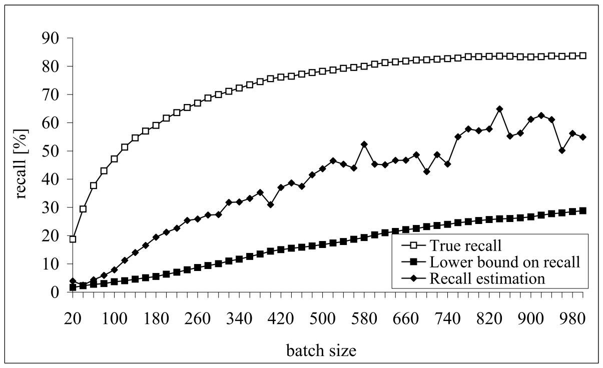

Figure 8. Recall using MUFIN

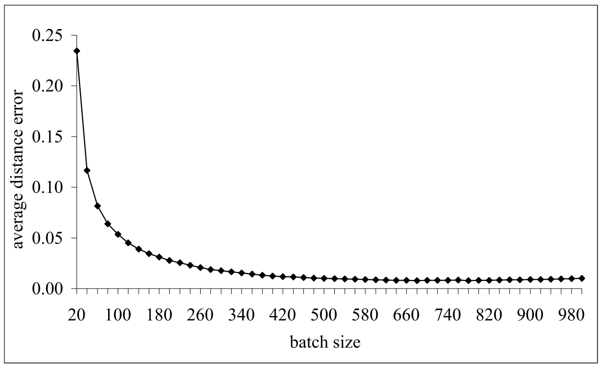

Figure 9. Distance error using MUFIN

This is better shown in Figure 9 as the *Average distance error*. This measure is computed as an average difference of distances (from query object) of objects in exact and approximate result. The average distance error lower than 0.05, which is achieved for $\epsilon > 100$, shows that the approximate and precise results are very similar. Thus, the images found by the approximation do not differ much, at least not according to the similarity expressed by the used descriptors. *Quality loss* graph in Figure 10 represents the

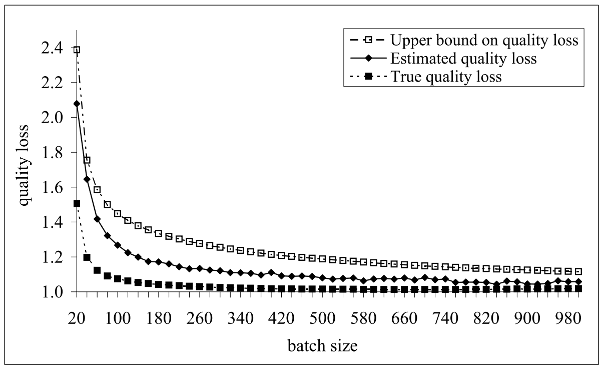

Figure 10. Loss of quality using MUFIN

measure described in Section 4. Similarly to the recall, the true quality loss can only be computed when the exact result is known (final threshold value is the distance of the worst object in the precise result). The graph also shows the estimated values that were guessed by the extrapolation and the upper bound that is known at the ϵ^{th} iteration. Quality loss of the precise result is equal to 1, so values near to 1 show a very good approximation result.

As we can see, all the other measures show that quality increases faster for lower values of ϵ. This can be used for optimization of query processing: On average, we can get as good as 80% recall in about 4 seconds for top 50 queries which is fairly acceptable for an user. To get better recall the system would need considerably more time. Since the notion of similarity is subjective in its own, this may not pay off.

6 Conclusion

We have described a distributed multi-overlay metric-based indexing network MUFIN. It can run any number of metric-indexing overlays in a peer-to-peer context that can be queried independently. On the peers, a local indexing structure such as M-Tree can be employed to further enhance the searching capabilities. By utilizing the zero-overlay, which is backed by a common distributed hash table, it supports both the sorted and random access necessary to fully utilize the threshold algorithm.

We have implemented our distributed variant of TA that also supports tunable approximation. When the system returns approximate result it should also give the user some information about its quality. Then the user can decide whether to run the query again with some other parameters or if the results are satisfactory. However, the quality cannot be computed without comparing the result to an exact result which is not available. Thus, we are predicting the threshold and maximal-distance values using extrapolation to give better estimation.

We have conducted experiments using a MUFIN instance loaded with 1.6 million images. Their visual content was obtained by MPEG-7 extraction techniques as five descriptors: three for colors, one for edges and one for texture, all of which are compared using a metric function also defined in MPEG-7. Thus, we have confirmed that the MUFIN instance can be employed to search the complex visual information.

However, our experiments also revealed that the estimation is still not very good as it is quite far from the true values. Thus, we would like to research this issue further. We would also like to explore the possibilities of re-ranking the results once they are delivered to the user using user's feedback and automatic tuning of the approximation.

7 Acknowledgments

This work has been supported by EU IST FP6 project 045128 (SAPIR), the national research project 1ET100300419, and the Czech Grant Agency project 201/07/P240. Especially, we are grateful to our SAPIR partners from ISTI-CNR, Pisa, who extracted the features for the images used in experiments.

References

[1] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic xml documents with distribution and replication. In *Proc. of the ACM SIGMOD'03*, pages 527–538, 2003.

[2] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. of the 24th ACM SIGIR'01*, pages 35–42, 2001.

[3] J. Aspnes and G. Shah. Skip graphs. In *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 2003.

[4] M. Batko, C. Gennaro, and P. Zezula. Similarity GRID for searching in metric spaces. In *DELOS Workshop: Digital Library Architectures, Lecture Notes in Computer Science*, volume 3664/2005, pages 25–44, 2005.

[5] M. Batko, D. Novak, F. Falchi, and P. Zezula. On scalability of the similarity search in the world of peers. In *Proceedings of INFOSCALE 2006, Hong Kong, May 30 – June 1, 2006*, pages 1–12, New York, NY, USA, 2006. ACM Press.

[6] B. Bustos and T. Skopal. Dynamic similarity search in multi-metric spaces. In *MIR '06: Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pages 137–146. ACM Press, 2006.

[7] S. Chaudhuri, L. Gravano, and M. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):992–1009, 2004.

[8] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. of the 20th ACM PODS'01*, pages 102–113, 2001.

[10] F. Falchi, C. Gennaro, and P. Zezula. A content-addressable network for similarity search in metric spaces. In *Proceedings of DBISP2P*, pages 126–137, 2005.

[11] U. Gntzer, W. Kieling, and W.-T. Balke. Towards efficient multi-feature queries in heterogeneous environments. In *ITCC '01: Proc. of the Int. Conference on Information Technology: Coding and Computing*, pages 622–628. IEEE, 2001.

[12] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[13] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proc. of VLDB'03*, pages 321–332, 2003.

[14] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In *Proc. of the 29th VLDB'03*, pages 129–140, 2003.

[15] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top-k query algorithms. In *Proc. of the 31st VLDB'05*, pages 637–648, 2005.

[16] MPEG-7. Multimedia content description interfaces. part 3: Visual. ISO/IEC 15938-3:2002, 2002.

[17] S. Nepal and M. V. Ramakrishna. Query processing issues in image(multimedia) databases. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, page 22. IEEE, 1999.

[18] D. Novak and P. Zezula. M-Chord: A scalable distributed similarity search structure. In *Proc. of 1st INFOSCALEi'06*, pages 1–10, New York, NY, USA, 2006. ACM Press.

[19] M. Theobald, R. Schenkel, and G. Weikum. The topx db&ir engine. In *ACM SIGMOD'07 Conference*, pages 1141–1143, 2007.

[20] C. Yu, P. Sharma, W. Meng, and Y. Qin. Database selection for processing k nearest neighbors queries in distributed environments. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 215–222, New York, NY, USA, 2001. ACM.

[21] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer-Verlag, 2006.

[22] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with m-trees. *The VLDB Journal*, 7(4):275–293, 1998.