A filter and refinement approach to RBIR

Michael Mlivoncic and Roger Weber

Institute of Information Systems, ETH Zentrum, 8092 Zurich, Switzerland
{mlivoncic, weber}@inf.ethz.ch

Abstract: A promising trend in content based image retrieval (CBIR) is the incorporation of the notion of objects into the similarity evaluation. Images are automatically segmented into a dynamic number of regions that roughly correspond to objects. In region based image retrieval (RBIR), we compute standard feature characteristic atop those segments and evaluate the similarity between images based on the similarity of its segments. This requires the application of matching techniques, like the Hungarian algorithm, that are prohibitivly expensive for collections beyond a few thousand images. In this paper, we provide a filter and refinement search algorithm to solve RBIR queries in reasonable times for even large image collections. We describe the bounding functions as well as implementational issues of our approach. Experimental results show that retrieval performance surpasses alternative approaches by a factor of 5.

1 Introduction

Content-based image retrieval (CBIR) systems have been proposed and deployed in the last few years (cf. [5]) in order to fulfill the need to find images that cover a certain information need of a user even in cases where no textual description of the content is available. Features, like color histograms express a certain characteristics of the images and can be used to assess the pairwise (dis)similarity of images. Given a reference image we can find and rank the most similar images from a collection according to the (dis)similarity measure. Many available systems use this "query by example" approach. However, one experiences a semantic gap between the low-level feature descriptions and the high level information need of a user, e.g., a user is looking for a red car and not a red house. To narrow this gap, recent systems (cf.[2]) extended CBIR technology towards region based image retrieval (RBIR). The notion of a region basically sketches a semantic object inside of an image. In an example query, a user would mark the prospective reference regions, e.g. the red car and its black tires. The system should come up with images containing most similar regions.

To enable RBIR queries, we segment images into regions using the JSEG algorithm (cf. [8]). From the set of homogeneous image regions, we extract local features for each region, e.g. color, texture and shape. Instead of one feature vector per image, we now have a varying number of feature vectors, each describing a region of an image. (Dis)similarity is again determined by the distance of the two images but as opposed to a CBIR system where this denotes the distance between two feature vectors, we have to compute the distance between two sets of regions. Therefore, we have to relate regions from the first set (i.e. the reference regions) to regions from the second set.

Existing RBIR systems like WALRUS [4] and Blobworld [2]) utilize index structures to find the matching regions, however they are looking for the best matching region from the collection for each reference region in isolation: For example, Blobworld [2] retrieves for each reference region a predefined number of similar regions from the database. Then, only images which appear at least once in these result sets are considered. Even though this approach is very efficient, it does not always reflect the desired semantics: Suppose, you are looking for an image with *two* cars. If the system matches the two regions containing the cars with the identical region of a car in the database, the corresponding image will obtain a good ranking. Images with multiple cars would not necessarily have the best rankings. To overcome this drawback, we use an 'exactly one' semantic to avoid that one region in the first set is matched with two regions in the second set.

Another interesting RBIR system is SIMPLIcity. It uses the IRM [7] measure to assign similarity scores between two region sets. IRM is based on the observation that segmentation algorithms tend to split objects into several regions. Obviously, when comparing these two images, we have to match the single region in

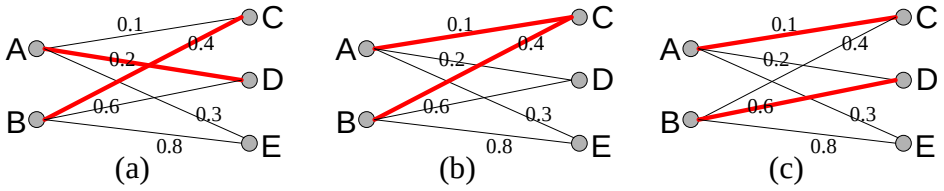


Figure 1: Optimal matching, lower and upper bound.

the first image with the two regions in the second image. IRM computes the similarity between a region A and two regions $\{B,C\}$ as a weighted sum of the similarities between A and B, and A and C. However, this heuristics leads to inaccurate values especially if the similarity measure takes shape, area, or location features into account. To solve this problem, we would have to merge the regions $\{B,C\}$, to compute features for this new partition, and to determine the similarity to A based on these new values. But computing features at query time is prohibitively expensive. On the other hand, storing features for all combinations of regions quickly becomes a storage problem and IO-operations negatively affect retrieval efficiency.

Matching all the regions from one set with the best fitting region from another set with the additional constraint that each target region should be matched at most once, is known as the minimal assignment problem. The matching is based on the distances between each of the single regions and has the property that each region of the reference set has exactly one corresponding region in the other set and the overall distance of the complete matching is minimal. This problem can be solved with the Hungarian algorithm [3]. However, on a huge image collection it is prohibitively expensive, as the algorithm has a complexity of $O(m^3)$ being m the number of regions to match. Our experiments have shown that this can take over 25 seconds for computing 100,000 matches with $m = 25$ using the Hungarian algorithm.

In this paper, we describe an efficient approach, based on the correct evaluation using the Hungarian algorithm but in corporation with a filter and refinement algorithm. We provide bounds on the distances between two images in order to filter out those images that are remote to the reference image and keep those that are likely to be close to a reference image. Those so called candidates are evaluated in the second phase. We will see that these bounds are correct and tight enough to filter with a high selectivity. Extensive experiments revealed that the execution times can be reduced by a factor of 5 due to the selectivity of the bounds.

This paper is structured as follows: Section 2 describes the general approach we have taken in RBIR. Section 3 describes the implementation of the data structure, DynRegFile, that allows for similarity searches using our algorithm. Section 4 describes some experiments and findings from our prototype implementation and Section 5 concludes the paper.

2 The filter and refinement approach

In CBIR, the (dis)similarity between two images is measured by taking the distance between their representing vectors in the given feature space. However, in RBIR we have to extend this approach, as every image consists of a variable number of regions and is therefore represented by a set of vectors. The distances between each region vector from one image to each region vector of the other image form a distance matrix. Corresponding regions are those that are similar, i.e. they have a low distance in feature space and therefore correspond to small values in the distance matrix. Our objective is to match corresponding regions from both sets one-by-one such that the overall distance between the two sets is minimal. The overall distance is computed as the average distance between the matched regions. This problem is widely known as the assignment problem and the Hungarian algorithm [3] efficiently determines its solution.

The assignment problem is usually depicted as a bipartite graph as in our example in Figure 1 a): We need a mapping of the regions A and B from one set to the regions C,D and E from another set. The distances of the matrix are given by the numbers along the edges between the nodes. The optimal matching is denoted by the bold edges. The overall distance is therefore $\frac{0.4+0.2}{2} = 0.3$. From the example, we can see that it is not sufficient to perform local optimizations: Matching A with C would be cheaper (than with D), but then we could not match B with C. As mentioned before, such minimal matching problems can be solved using

the Hungarian algorithm [3]. Based on this matching strategy, the **SCAN** algorithm iterates through all images of the collection and evaluates the overall distance for each image using the Hungarian algorithm. It keeps track of the best matches and returns them after the scan. However, the execution costs are prohibitive for larger collections. Therefore we use a different method where we avoid a lot of expensive Hungarian computations in order to save on execution costs. The basic idea is as follows: We determine (lower and upper) bounds for the optimal matching distance and keep track of those images, that are likely to have the lowest distances towards the reference images according to their bounds. In a second step, we determine the exact distances using the Hungarian approach and return the result set of the best matching images. Those bounds have to be markedly computationally cheaper compared to the Hungarian matching and tight enough to filter out a large fraction of the collection. In the following, we describe such a two-phase filter and refinement algorithm. For the sake of presentation, we will describe the algorithm for the case of $k = 1$, i.e. we are just interested in the most similar image of the collection. For $k > 1$ the extensions are straightforward. The phases of the algorithm are defined as follows:

I.) Filter step: Iterate over all images of the collection and perform the following steps for each image: 1.) determine the lower bound distance to the reference regions. 2.) if the current lower bound is smaller than the lowest upper bound encountered so far, the image is preserved as a candidate, otherwise it is discarded. 3.) In case of a new candidate: Also determine and store its upper bound.

II.) Refinement step: Access the candidates that remain after the filter phase in ascending order of their lower bounds: 1.) Compute the exact distance towards the reference regions using the Hungarian algorithm. 2.) keep the best (i.e. smallest) distance computed so far together with the corresponding image (oid). 3.) as soon as the current lower bound exceeds the best encountered distance so far, we can (early) abort and return the best seen image.

In order to compute a lower bound we match each region on the left side with the closest (by distance) region on the right side. In the example (Figure 1 b)), we can therefore match region A with region C, as well as region B. This yields in a lower bound of 0.25. As the Hungarian matching has to obey the "one-by-one" constraint in addition to that, this matching is definitely a lower bound to the problem and clearly easy to compute with costs of at most $O(m^2)$. An upper bound to the Hungarian distance also assigns the best fitting region of the target set but excludes that region from the target set afterwards to avoid further matching. Obviously such a matching is non-optimal and therefore it is clearly an upper bound to the problem. Such an upper-bound maching with a distance of 0.35 is depicted in Figure 1 c) for our example.

3 The Dynamic Regions File

Our search algorithm described above requires an iterator over the vectors of the images in the first phase. It is crucial for a good performance of our retrieval system to store the feature vectors in a format where they can be efficently accessed. Since the number of feature vectors of the images differ, the data structure has to handle entries of dynamic length. In some cases, one wants to combine several features,e.g. color and texture, in a specific query. In this scenario, each single vector file should be organized such that the feature values of a single image can be found at the same position in each of the files in order to allow parallel access to all feature values of an image during the sequential access phase [1]. A sequential file where we append each new object's feature vector to the file would be sufficient, but we also want to allow for deletes of objects and later reuse of space by inserts. Our basic approach is to store region vectors of an image in consecutive slots of the size of a region vector. For the delete and update operations, we need to know the slots that are allocated for a certain object. Therefore, we need a mapping that provides us the first slot number assigned to an object given its OID.

In the following, we introduce the "Dynamic Regions File" (DynRegFile), that stores region based feature vectors of dynamic length.

Organization of the File: The DynRegFile is composed of slots of *fixed* size. Each such slot stores an OID plus the feature vector of *one* region of the image with the given OID. Note that the feature vector of each single region has a fixed length. The features of an image are stored as a group of region vectors in consecutive slots of the DynRegFile; all of them having the same OID. On the request of a certain images' feature vector, it is constructed "on the fly" by gluing together all subsequent region vectors with the same

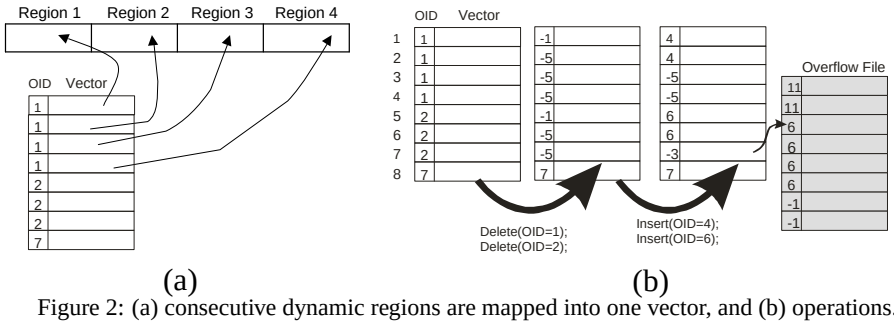


Figure 2: (a) consecutive dynamic regions are mapped into one vector, and (b) operations.

OID. Conversely, on adding an images' vector it is splitted into a vector group and the resulting region vectors are written to the assigned slots. In Figure 2 a), a request for OID 1 is shown. The order of the regions depends on the output from the segmenter: The first region is assigned to the lowest slot in the file and so on.

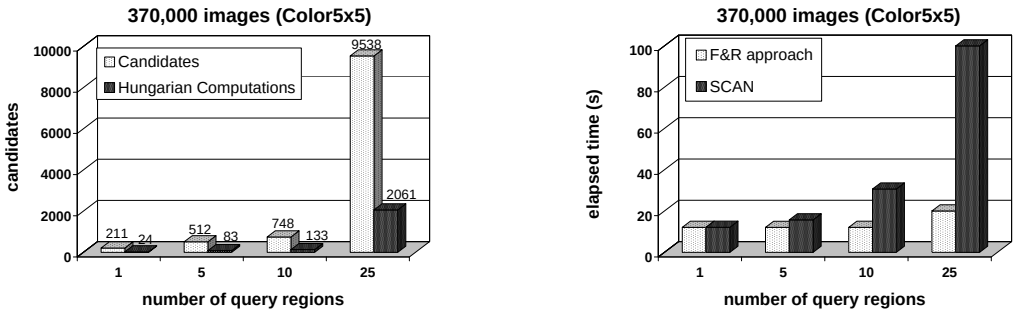
Delete and insert operations: A delete operation is performed by invalidating the respective vectors: The first slot of the object in the DynRegFile is marked with $OID_{UNKNOWN}$ (-1) and all consecutive slots of the same object are marked as $OID_{FREEREGION}$ (-5). By doing so, we can easily distinguish between two adjacent invalidated vector groups, as each group starts with an $OID_{UNKNOWN}$. This is important as we always need at least one slot per oid.

Insert operations should reuse those holes in the file produced by preceding delete operations in order to keep the data structure compact and efficient. Region vectors of the new object are filled into the first free slots in the DynRegFile. In case where the new image has less regions than the image residing in the file at that location before, the unused vector slots are marked as $OID_{FREEREGION}$. But in the case of an increased number of regions, obviously not all vectors fit into the old slots. To support such inserts without having to reorganize the file each time, we use an overflow file. Now, we fill all old slots in the DynRegFile except the last one. This slot is filled with a reference to the overflow file with an OID of $OID_{OVERFLOW}$ (-3). The vector data field of this slot is used to encode a reference to the position where the next part of the vector group can be found within the overflow file. Then the overflow file is filled with the remaining vectors of the group starting at that position. Figure 2 b) gives an example of such an insert. We restrict the overflow file to a rather small size as this file should always reside in main memory and act as a buffer. If the overflow file exceeds its maximum size during an insert operation, a new instance of the DynRegFile is created in the background with all region vectors of an object residing in consecutive slots again. Meanwhile, incoming queries can be served by the "old" file and overflow file while new inserts can still be logged at the end of the overflow file.

Accessing the DynRegFile: On the interface level, the Iterator class provides transparent access to the whole vector group of the images' region vectors: The iterator has to read the DynRegFile slot by slot until the next entry has a different OID (including $OID_{UNKNOWN}$,$OID_{FREEREGION}$ or $OID_{OVERFLOW}$) In case of $OID_{OVERFLOW}$, it continues reading the corresponding vectors of the group from the overflow file. Having read all vectors of a group, it builds the images' feature vector and returns it together with the number of regions of the current image. Besides that, with each step in the iteration process, the mapping of external to internal positions is updated in order to support random accesses after a scan.

4 Experiments

We performed several experiments with our C++ prototype implementation on an Windows 2000 Server/ Intel Pentium 4 platform with 1.8 GHz, 256 MB main memory, and a hard disk with a throughput of 25 MB/s. From a collection of 19,000 images we extracted 9 regions on average using the JSEG algorithm and from a set of 370,000 images we extracted 5x5 overlapping regions per image. Each region was described by a color moment feature [6] with 9 dimensions. For brevity reasons, we present only some of the experiments with the larger data set in the following. Our observations also hold for the other experimental settings and parameters not presented here.



(a) (b)

Figure 3: Candidates and execution times of the algorithms

Figure 3 shows results for that data set. We varied the number of reference regions from 1 to 25 (x-axis). The first observation is that our bounds are sufficient tight: Figure 3 a) presents the absolute numbers for the candidates that remain after the first phase. For 25 regions, there are about 9500 candidates ($\approx 2.6\%$) remaining from the collection of 370,000 images. In the second phase, we only have to do about 2000 Hungarian computations ($\approx 0.5\%$) before the algorithm early aborts. As the execution of the algorithm is CPU bounded, reducing the number of Hungarian computations by a factor of 180 is directly reflected in the overall execution times (cf. Figure 3 b)): While the *SCAN*-algorithm needs about 100 seconds for a query with 25 reference regions, the *Filter and Refine*- approach reduces this by a factor of 5 to about 20 seconds.

5 Conclusion

In this paper, we described an approach for an efficient evaluation of RBIR queries. We provided bound functions together with a Filter-and-Refinement algorithm that can be used to reduce computational costs by filtering out a large number of images from the whole collection before performing the costly similarity computations on the remaining candidates. Our experimental results show that we can thereby improve the overall performance by a factor of 5 due to a selectivity of 0.5%. Furthermore, we described the DynRegFile-data structure suitable to accommodate the variable length feature data of the images and allow efficient access for the algorithm described.

References

[1] K. Böhm, M. Mlivoncic, H.-J. Schek, and R. Weber. Fast Evaluation Techniques for Complex Similarity Queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Rome, 2001.

[2] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A System for Region-Based Image Indexing and Retrieval. In *Visual Information and Information Systems, Third International Conference,VISUAL '99*, volume 1614 of *Lecture Notes in Computer Science*, p. 509–516, Amsterdam, The Netherlands, 1999.

[3] D. E. Knuth. *The Stanford Graph Base*, p. 77–89. ACM press, New York, 1993.

[4] A. Natsev, R. Rastogi, and K. Shim. WALRUS: a similarity retrieval algorithm for image databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 28(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, p. 395–406, Philadelphia, PA, USA, June 1–3, 1999, 1999.

[5] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. H. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC Project: Querying Images by Content, Using Color, Texture, and Shape. In *Storage and Retrieval for Image and Video Databases*, volume 1908 of *SPIE Proceedings*, p. 173–187, San Jose, CA, USA, 1993.

[6] M. A. Stricker and M. Orengo. Similarity of Color Images. In *Storage and Retrieval for Image and Video Databases (SPIE)*, volume 2420 of *SPIE Proceedings*, p. 381–392, San Diego/La Jolla, CA, USA, 1995.

[7] J. Z. Wang. SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 23, p. 947–963, 2001.

[8] B. M. Y. Deng and H. Shin. Color image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1999.