Efficient Region-Based Image Retrieval

Roger Weber
Database Research Group
Swiss Federal Institute of Technology (ETH)
ETH Zentrum, CH–8092 Zurich, Switzerland
weber@inf.ethz.ch

Michael Mlivoncic
Database Research Group
Swiss Federal Institute of Technology (ETH)
ETH Zentrum, CH–8092 Zurich, Switzerland
mlivoncic@inf.ethz.ch

ABSTRACT

Region-based image retrieval (RBIR) was recently proposed as an extension of content-based image retrieval (CBIR). An RBIR system automatically segments images into a variable number of regions, and extracts for each region a set of features. Then, a dissimilarity function determines the distance between a database image and a set of reference regions. Unfortunately, the large evaluation costs of the dissimilarity function are restricting RBIR to relatively small databases. In this paper, we apply a multi-step approach to enable region-based techniques for large image collections. We provide cheap lower and upper bounding distance functions for a recently proposed dissimilarity measure. As our experiments show, these bounding functions are so tight, that we have to evaluate the expensive distance function for less than 0.5% of the images. For a typical image database with more than 370,000 images, our multi-step approach improved retrieval performance by a factor of more than 5 compared to the currently fastest methods.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process, Information filtering

General Terms

Algorithms, Performance

Keywords

region-based image retrieval, RBIR, CBIR

1. INTRODUCTION

An increasing number of applications require sophisticated image search facilities. The most prominent ones are medical imagery, remote sensing, internet, news and media. Traditionally, these applications mainly deploy keyword-based search operations. Newer search engines additionally integrate content descriptors derived from the pixel data of the images into their retrieval algorithms [16]. Often, one refers to this approach as content-based image retrieval (CBIR). Region-based image retrieval (RBIR) [5, 15, 19] is a promising extension of the classical CBIR: rather than deploying global features over the entire content, RBIR systems partition an image into a number of homogenous regions and extract local features for each region. Apart of the classical similarity search, RBIR provides novel query types: most importantly, the search for images containing similar parts of a reference image, e.g., *"find pictures of this brown dog (represented as a region) in an arbitrary environment"*. Searching with a single reference region is a simple extension of CBIR. But the more interesting case of searching with multiple reference regions is less obvious to solve, and much more expensive as we will see.

In this paper, we address the problem of *efficiently* and *effectively* determining relevant images in a database given a set of reference regions. As a preliminary step, we have to define the dissimilarity (distance) between two images, i.e., between two sets of regions. We have selected the approach of the Windsurf system [2] since it clearly outperforms other systems from an effectiveness point of view. The approach of the Windsurf system is to match each region from the first set with exactly one region of the second set. Given such a matching, it determines the overall dissimilarity as the average distance between matched regions. Among the set of possible matchings, the Windsurf system chooses the one leading to a minimal overall distance. This optimization problem is a variant of the well-known Assignment Problem. Using the Hungarian Algorithm [11, 13], the complexity of distance evaluations for two region sets is $O(r^3)$ with r being the number of regions in both sets.

However, the Winsurf system was not able to implement an *efficient* search algorithm to identify the most similar images according to their effective similarity measure. As the result in [2] illustrate, the proposed search algorithm does not perform significantly better than a brute-force search (referred to as *SCAN*) that computes distances for all objects in the database. In this situation, it is obvious that the large costs of the Hungarian Algorithm impair search efficiency of RBIR queries for already a small number of reference regions. Even worse, newer approaches extract hundreds of tiny regions called *"points of interests"* from an image [10]. These points correspond to characteristic 1-pixel regions of the image. With distance evaluation costs of $O(r^3)$, this promising approach is restricted to small databases.

The main contribution of this paper is an efficient solution for multi-region searches as proposed by the Windsurf system. For that purpose, we applied the multi-step paradigm [1, 9, 14, 17, 20] to region-based retrieval: in a first step (filtering), a lower (and upper) bounding distance function on the dissimilarity measure determines a set of candidates. Then, a second step (refinement), identifies the correct result set by computing exact distances only for these candidates. In the course of this work, we provide cheap bounding functions for our region-based dissimilarity measure. We further show with experiments for a typical image application that these functions are sufficiently tight such that less than 0.5% of the images remain as candidates after the first step. The performance comparisons reveal that the multi-step approach reduces search costs by a factor of 5.

The paper is structured as follows: the next section reviews related work. Section 3 defines the underlying dissimilarity function for multi-region queries. Section 4 derives lower and upper bounding functions, and proposes two multi-step algorithms. Section 5 discusses performance characteristics of our solution, and Section 6 concludes.

2. RELATED WORK

Blobworld [5], an early RBIR system, mainly focused on the segmentation of images into homogeneous regions from a color and texture perspective. During query evaluation, Blobworld retrieves for each query blob a number of similar database blobs using a high-dimensional index structure. Only images which appear at least once in these result sets are considered: for each image, the score is given by a fuzzy combination over the scores between the query blobs and their most similar blob in that image. Although Blobworld demonstrated the potential of RBIR systems, their query model and retrieval algorithm does not satisfy our requirements: 1) With Blobworld, it is possible that an image blob is matched with several query blobs. 2) Query evaluation, as shown in [2], leads to missing the correct best images.

WALRUS [15] segments images using wavelets. The usage of wavelets for segmentation resulted in good segmentation quality. In the querying phase, WALRUS retrieves for each query region all database regions having a distance smaller than ε. Then WALRUS sums up the sizes of the retrieved regions per image and returns only images that exceed a threshold on the matched region sizes. However, from a retrieval perspective several open issues remain: 1) vague definition of the similarity function for multi-region queries, and 2) the evaluation requires hard-to-guess thresholds.

SIMPLIcity uses IRM [19] to assign similarity scores between two region sets. IRM is based on the observation that segmentation algorithms tend to split objects into several regions. For instance, a dog in a first image is represented by a single region while a dog in a second image is split into two regions. Obviously, when comparing these two images, we have to match the single region in the first image with the two regions in the second image. To avoid such situations, we demand for better segmentation algorithms. In fact, the work of Hermes et al. already demonstrated an astonishing improvement of segmentation quality [12] compared to earlier solutions. Given an effective segmentation algorithm, we no longer have to take many-to-many relations between regions of two sets into account.

The similarity measure of this work is based on the Windsurf system [2]. Windsurf describes the matching as an Assignment Problem and solves this optimization problem with the Hungarian Algorithm. Further, the authors of [2] present the $\mathcal{A}_0^{\mathcal{WS}}$-algorithm: for each query region, they open a stream for the database regions ordered by their similarities to the query region. The fetching phase ends, if there exists a complete matching for an image. Then, a random access phase computes the similarities of the images for which the fetching phase has seen at least one of their regions. While the $\mathcal{A}_0^{\mathcal{WS}}$-algorithm provably evaluates the exact result (in contrast to the algorithm of Blobworld), it performs only marginally better than a brute-force method. In some of the experiments, the performance was even worse.

3. MULTI-REGION IMAGE SEARCH

We have selected the dissimilarity function of the Windsurf system due to its intuitive character and good search quality [2]. Furthermore, this measure solves a number of essential problems simpler approaches suffer from, e.g., the two tiger problem of Blobworld[1]. In the following, we describe our extension of the retrieval model of Windsurf (distinction of query types with the introduction of penalty values). Furthermore, we assume that the automatic segmentation algorithm (e.g., [7, 12]) partitions an image into homogenous regions, i.e., the segmentation quality is satisfactory accurate.

3.1 Region-Based Dissimilarity

Let $\mathcal{R}^Q = \{R_1^Q, \ldots, R_m^Q\}$ be the set of query regions, and $\mathcal{R}^{O_l} = \{R_1^{O_l}, \ldots, R_{n_l}^{O_l}\}$ the set of regions for the l-th image in the database $\mathbb{O}$, $1 \le l \le N$. A region R can be described by any means, e.g., multi-dimensional features. We only demand for a function $\delta(R_1, R_2)$ that measures the proximity between two regions R_1 and R_2. Then, the distances between regions of two region sets $\mathcal{R}^1$ and $\mathcal{R}^2$ span a matrix

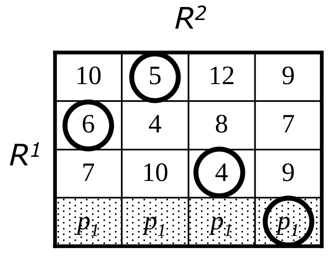
$$\mathbb{D} = \left\{\delta(R_i^1, R_j^2)\right\}_{i,j} \in \mathbb{R}^{|\mathcal{R}^1| \times |\mathcal{R}^2|} \tag{3.1}$$

Its rows and columns correspond to regions from $\mathcal{R}^1$ and $\mathcal{R}^2$, respectively. Figure 1 (a) shows an example for two region sets $\mathcal{R}^1$ and $\mathcal{R}^2$ containing 3 and 4 regions, respectively (ignore the last row for now).

From another perspective, consider the regions of the two sets as nodes of a bipartite graph and distances as weights on edges between these nodes (cf. Figure 1 (b)). To determine an overall distance, we must assign regions from the first set to regions from the second set. Obviously, this matching of regions should contain only pairs of similar regions and should comprise as many region pairs as possible. In our model and in contrast to IRM [19], each region can match with at most one region from the other set[2]. We introduce the notion of a *complete matching*: a matching $\mathbb{M} \subseteq \mathcal{R}^1 \times \mathcal{R}^2$ is complete, if it is maximal and does not contain two edges

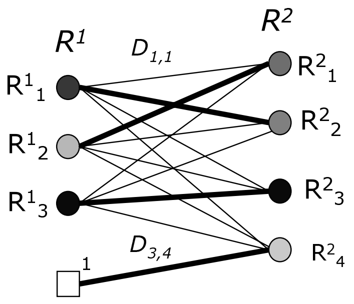
[1]The two tiger problem is as follows: a query with two tigers should not match images in the database depicting just one tiger. However, Blobworld's and WALRUS' definition of similarity do not take this into account. As a result, these systems may report images with one tiger at higher ranks than images with two tigers.

[2]This is because newer segmentation algorithms seldom divide objects into many regions [12].

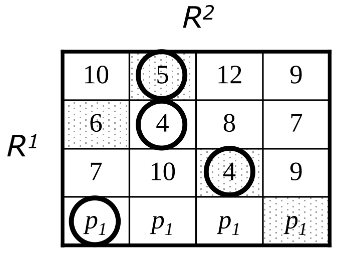


R^2
R^1
10 5 12 9
6 4 8 7
7 10 4 9
p_1 p_1 p_1 p_1

(a) distance matrix **D**



(b) matching in bipartite graph



R^2
R^1
10 5 12 9
6 4 8 7
7 10 4 9
p_1 p_1 p_1 p_1

(c) lower and upper bounds

Figure 1: Distance matrix, matching, and optimal matching.

connected to the same region, i.e., each region appears at most once in $\mathbb{M}$. The dissimilarity measure $\delta(\mathcal{R}_1, \mathcal{R}_2)$ is then the average over the elements of the distance matrix $\mathbb{D}$ whose corresponding regions are represented by a tuple in $\mathbb{M}$. Recall that we demand for a maximal matching, i.e., the cardinality of $\mathbb{M}$ is given by $\min\left(|\mathcal{R}^1|, |\mathcal{R}^2|\right)$.

Figure 1 (a) and (b) depict a complete matching with bold edges in the bipartite graph and circles around the corresponding matrix elements (ignore the last row in $\mathbb{D}$ and the special box-node in the bipartite graph). The distance for this matching is $(5 + 6 + 4)/3 = 5$. Notice that node R_4^2 is not matched with any node in $\mathcal{R}^1$. For some query types, we want to penalize such non-matches. To this end and as an extension of the Windsurf model, we introduce *"null regions"* which match with any region not contained by $\mathbb{M}$. Furthermore, the distance to a null region is constant, i.e., the penalty for a non-match. Let $\square^1$ and $\square^2$ be the null regions for region sets $\mathcal{R}^1$ and $\mathcal{R}^2$, respectively. Then: $\forall R_j^2 : \delta(\square^1, R_j^2) = p_1$ and $\forall R_i^1 : \delta(R_i^1, \square^2) = p_2$. Now, we add so many null regions to the smaller region set until both sets have equal size (cf. Figure 1 (b)). As a consequence, the distance matrix $\mathbb{D}$ of $\mathcal{R}^1$ and $\mathcal{R}^2$ becomes quadratic: the new rows (insertions of $\square^1$) are filled with penalty p_1 (cf. Figure 1 (a), last row), and the new columns (insertions of $\square^2$) contain penalty p_2. Due to completeness, the distance function now selects from each row and each column exactly one value to determine the overall distance.

So far, we have not discussed how to match regions from the two sets. Obviously, we want to match the most similar regions. However, this is generally not feasible due to the completeness constraint. Instead, we optimize the matching as a whole:

$$\delta(\mathcal{R}^1, \mathcal{R}^2) = \min_{\substack{\mathbb{M} \subseteq \mathcal{R}^1 \times \mathcal{R}^2 \\ \mathbb{M} \text{ is complete}}} \varphi(\mathbb{M}, \mathbb{D}) \tag{3.2}$$

$$\text{with } \varphi(\mathbb{M}, \mathbb{D}) = \frac{1}{|\mathbb{M}|} \sum_{(R_i^1, R_j^2) \in \mathbb{M}} D_{i,j}$$

Note that Equation (3.1) formulates a variation of the well-known Assignment Problem [11]. The Hungarian Algorithm [11, 13] finds the optimal solution for a distance matrix $\mathbb{D} \in \mathbb{R}^{r \times r}$ in $O(r^3)$. Applied to the example in Figure 1, we obtain a distance of $(5 + 6 + 4 + p_1)/4$, i.e., the depicted matching is optimal.

3.2 Nearest Neighbor Retrieval

In the following, we only consider k nearest neighbor searches explicitly as they are more natural than range searches that require users to specify hard-to-guess thresholds to limit the result size. These restrictions are not limiting our approach as adapting the algorithms to also support range queries is straightforward. A first exact algorithm to find the k best matches is a brute-force scan:

Algorithm 3.1 (*SCAN*) *For each image stored in the database, compute the distance matrix for the set of reference regions and the set of image regions. Determine the overall distance to the query by solving Equation (3.1) with the Hungarian Algorithm. Return the k images having the smallest overall distances.*

The main problem of *SCAN* origins in the large number of invocations of the costly Hungarian Algorithm. With a database of N images and r regions per image, the complexity for searching with r reference regions is $O(Nr^3)$. Obviously, this leads to unacceptably large retrieval costs for already medium size databases. Moreover, newer approaches extract not only a few regions from an image, but hundreds of points of interest [10]. A brute-force search would limit this promising technique to small database sizes.

3.3 Query Types

Apart of the classical similarity search, our dissimilarity function enables further query types.

▶ **contains:** A user searches for images that *contain* a similar region for each reference region. Clearly, the user does not care if the image has further regions. Hence, we must not penalize non-matches of image regions, but we have to penalize non-matches of query regions. This leads to the following penalties: $\delta(\square^1, R_j^Q) = p_1 = 0$ and $\delta(R_i^Q, \square^2) = p_2 > 0$.

▶ **similarity:** This is the classical search type also known from CBIR systems. The user selects an image and looks for images in the database that have a similar number of similar regions. In this case, we set both penalties to the same value to ensure a symmetric distance function. Choosing a large penalty value leads to images having the same number of regions as the reference image.

► **part-of:** In analogy to the *contains* query, we set $p_1 > 0$ and $p_2 = 0$. Hence, we are looking for images whose regions have a similar counterpart in the reference image (part of the reference image).

For the similarity case, there already exists a large number of approaches that support this query type. This is not the case for containment and part-of queries. Further note that for containment and part-of queries, the distance function is not symmetric and the triangle inequality does not hold. This makes indexing of images for containment and part-of queries very difficult.

4. MULTI-STEP RBIR SEARCH

Although a lot of multi-dimensional index structures exists, they are not directly applicable to our RBIR query types as they are not capable of dealing with image descriptions that contain sets of feature vectors. In Blobworld [5] and WALRUS [15], each region is treated as a distinct object and all region features are stored in a multi-dimensional indexing structure like the X-Tree [3], or the VA-File [20]. For each reference region, the index is searched and among all the objects corresponding to a returned region, the overall best image is selected. However, this is not acceptable as the result is not correct in the general case (see [2] for a proof). The Windsurf system overcomes this problem with the $\mathcal{A}_0^{\mathcal{WS}}$-algorithm [2]. However, the reported performance is only marginally better than the one of *SCAN*.

A further candidate structure among many others is the M-Tree [6] that maintains objects based only on distances. The M-Tree requires a distance function that fulfills the properties of a metric. However, with our similarity definition, the distance function is not always a metric (containment and part-of queries). Furthermore, the triangle inequality does not hold for all query types. Consequently, we can only deploy these methods for similarity searches, but not for the entire range of RBIR queries.

In our approach, we have applied the multi-step search paradigm to reduce response times for RBIR queries. In the following, we describe how to derive cheap but tight bounding functions for Equation (3.1). Furthermore, we have extended the generic multi-step algorithm of [17] to optimize CPU and IO costs.

4.1 Preliminaries on Multi-Step Algorithms

Earlier work reports on the successful application of multi-step algorithms for distance-based search problems like, for instance, medical imagery [14], subsequence matching [9], or color histogram similarity search [1]. But also signature files [8] and the VA-File [20] follow the same principle: in a first step, an approximate function eliminates objects from the search. The remaining objects are called the *candidates*. Only for these candidates, the more expensive query test is evaluated. For distance-based search problems, we introduce a lower bounding distance function δ_{lb} for a distance function δ such that: $\forall Q, O \in \Omega : \delta_{lb}(Q, O) \leq \delta(Q, O)$. Assume we search for the k best matches for a query Q. Further, let $\delta^k(Q)$ denote the k-th smallest distance between Q and the objects in $\mathbb{O}$. Then, a lower bounding function δ_{lb} defines a set of candidates $\mathbb{C}_{lb}(Q)$ as follows:

$$\mathbb{C}_{lb}(Q) = \Big\{ O \,|\, O \in \mathbb{O} \;\wedge\; \delta_{lb}(Q, O) \leq \delta^k(Q) \Big\} \tag{4.1}$$

Note that if $\delta_{lb}(Q, O) > \delta^k(Q)$, object O cannot be among the k nearest neighbors. This leads to the multi-step paradigm: in a first step, we compute a set of candidates $\mathbb{C}$ by only considering lower bounds. In a second step, we identify the result set by determining exact distances for objects $O \in \mathbb{C}$. Correctness of this approach holds if the set of candidates $\mathbb{C}$ contains at least all lower bound candidates, i.e., $\mathbb{C} \supseteq \mathbb{C}_{lb}(Q)$ [17]. In this context, δ_{lb}-optimality is given if the set of candidates comprises all and only the objects in $\mathbb{C}_{lb}(Q)$. But how to compute $\mathbb{C}_{lb}(Q)$ without knowing $\delta^k(Q)$ in advance? Seidl et al. [17] solved this problem by accessing candidates from a multi-dimensional index structure in increasing order of their lower bounding distances. In our scenario, however, this is not feasible.

4.2 Lower and Upper Bounds for RBIR

Recall, that an optimal matching $\mathbb{M}^{\circledast}$ for Equation (3.1) selects exactly one element from each row and from each column of the distance matrix $\mathbb{D}$. The distance $\delta(\mathcal{R}^1, \mathcal{R}^2)$ is then the average over the selected values. We can lower bound this distance by averaging over the minimal value of each row in $\mathbb{D}$:

$$\delta_{lb}(\mathcal{R}^1, \mathcal{R}^2) = \varphi(\mathbb{M}_{lb}, \mathbb{D}) \quad \text{with} \tag{4.2}$$

$$\mathbb{M}_{lb} = \left\{ (R_i^1, R_j^2) \,|\, R_i^1 \in \mathcal{R}^1 \wedge R_j^2 = \arg\min_{R_{j'}^2 \in \mathcal{R}^2} D_{i,j'} \right\}$$

In Figure 1 (c), circles mark the selected elements for the lower bound of the illustrated example. The lower bound is given as $(5 + 4 + 4 + p_1)/4$ which is smaller than the exact distance given by $(5 + 6 + 4 + p_1)/4$. However, we observe that the matching is not complete, i.e., R_2^2 is matched with regions R_1^1 and R_2^1.

The algorithm of [17] only requires a lower bounding function. For our extension, however, we further need an upper bounding function. δ_{ub} is an upper bounding distance function for δ if: $\forall Q, O \in \Omega : \delta(Q, O) \leq \delta_{ub}(Q, O)$. To determine an upper bound for $\delta(\mathcal{R}^1, \mathcal{R}^2)$, we can simply take an arbitrary complete matching. However, we want to have a *tight* upper bounding function. Therefore, we select from each row the minimal value and discard the corresponding column for the remaining rows:

$$\delta_{ub}(\mathcal{R}^1, \mathcal{R}^2) = \varphi(\mathbb{M}_{ub}, \mathbb{D}) \tag{4.3}$$

$$\text{with} \;\; \mathbb{M}_{ub} = \left\{ (R_i^1, R_j^2) \,|\, R_i^1 \in \mathcal{R}^1 \;\; R_j^2 = \arg\min_{R_{j'}^2} D_{i,j'} \right\}$$

$$\text{and} \;\; R_{j'}^2 \in \left\{ R_{j^*}^2 | R_{j^*}^2 \in \mathcal{R}^2 \wedge \nexists i^* < i : (R_{i^*}^1, R_{j^*}^2) \in \mathbb{M}_{ub} \right\}$$

The shaded cells in Figure 1 (c) contain the selected elements for the upper bound. It is given as $(5 + 6 + 4 + p_1)/4$ which is equal to the correct distance (of course, this does not generally hold).

With $\mathbb{D} \in \mathbb{R}^{r \times r}$, i.e., r regions per set, the complexity to determine bounds is only $O(r^2)$ instead of $O(r^3)$ when using the Hungarian Algorithm. Hence, the bounds are significantly cheaper from a complexity perspective. Note that, in any case, we first have to compute the distance matrix; the complexity of this operation is given by $O(r^2)$. Figure 2(a) shows CPU costs for the Hungarian Algorithm, bound computations and distance matrix evaluations as a function of the number of regions (both region sets have equal size). A distance between two regions was computed as the Manhattan distance between two 9-dimensional vectors. For each measurement, we performed 100,000 evaluations. From the graphs, we derive that bounds are up to a factor of 40 cheaper than exact distances. However, we have to add the costs for the computation of the distance matrix. Hence, for 50 regions, we obtain CPU costs of 80 seconds for exact distances, and 16 seconds for the bounds. Hence, bound evaluations suffer from the high costs of distance computations. But still, bounds are by a factor of 5 cheaper then exact distances.

4.3 Using Only Lower Bounds

The algorithm of [17] accesses objects from an index structure sorted by their lower bounds. But for our lower bounds, there exists no appropriate index structure. Thus, we must adapt their algorithm slightly by introducing a priority queue (implemented as a heap) for objects according to their lower bounds:

Algorithm 4.1 (Only Lower Bound (*LOWBND*))

```
// First Step
// Let heap be increasingly ordered
iterate over $O \in \mathbb{O}$
  determine $\delta_{lb}(Q,O)$
  $heap$.push($\delta_{lb}(Q,O), oid(O)$)

// Second Step
// Let result be a list increasingly ordered on distances
$\delta^k = \infty$
repeat
  $(l,i)$ = $heap$.pop()
  exit if $l \geq \delta^k$
  read $O$ from database for oid $i$ and determine $\delta(Q,O)$
  $result$.insert($\delta(Q,O), oid(O)$)
  if $result$.getNumber()>$k$ then $\delta^k$ = $result$.getDist($k$)
until $heap$.isEmpty()
```

Note that multi-dimensional indexes as used in [17] also maintain a heap for the objects encountered during the search. There is even evidence that, with high-dimensional features, *LOWBND* performs comparable to the algorithm of [17] due to the inherent difficulties of searching such spaces [4, 20].

LOWBND experiences two problems: 1) the size of the heap is equal to the number of objects in the database. The complexity for maintaining this heap is $O(NlogN)$. 2) Candidate objects are re-visited during the second step. These visits impose random accesses on the hard disk to retrieve the object data. Apart of disk costs, the second step has to recompute the distance matrix for each candidate object.

4.4 Using Lower and Upper Bounds

Let $\delta^k_{ub}(Q)$ be the k-th smallest upper bound of the objects in the database. Then, an object O with: $\delta_{lb}(Q,O) > \delta^k_{ub}(Q)$ cannot be among the k nearest neighbors due to $\delta^k(Q) \leq \delta^k_{ub}(Q) < \delta_{lb}(Q,O) \leq \delta(Q,O)$. In other words, we obtain a second (larger) set of upper bound candidates:

$$\mathbb{C}_{ub}(Q) = \left\{ O | O \in \mathbb{O} \wedge \delta_{lb}(Q,O) \leq \delta^k_{ub}(Q) \right\} \supseteq \mathbb{C}_{lb}(Q) \tag{4.4}$$

To minimize heap maintenance costs in algorithm *LOWBND*, we insert objects into the heap only if they are upper bound candidates. If $\mathbb{C}_{ub}(Q)$ is markedly smaller than $\mathbb{O}$, we can save maintenance costs as the complexity is now reduced to $O(|\mathbb{C}_{ub}(Q)| \cdot \log(|\mathbb{C}_{ub}(Q)|))$. Furthermore, if $|\mathbb{C}_{ub}(Q)|$ is small, we can cache candidate objects read in the first step in main memory and access this cache when revisiting the candidates in the second step. For region-based search, we cache the distance matrix and thereby even avoid its recomputation in the second step. These considerations lead to a novel multi-step approach:

Algorithm 4.2 (With Upper Bound (*UPPBND*))

```
// First Step
// Let heap be increasingly ordered on lower bounds
// Let ub be a list increasingly ordered on upper bounds
$\delta^k_{ub} = \infty$
iterate over $O \in \mathbb{O}$
  determine $\delta_{lb}(Q,O)$
  if $\delta_{lb}(Q,O) \leq \delta^k_{ub}$ then
    $heap$.push($\delta_{lb}(Q,O), O$)
    determine $\delta_{ub}(Q,O)$
    $ub$.insert($\delta_{ub}(Q,O)$)
    if $ub$.getNumber()>$k$ then $\delta^k_{ub}$ = $ub$.getDist($k$)

// Second Step
// Let result be a list increasingly ordered
$\delta^k = \infty$
repeat
  $(l,O)$ = $heap$.pop()
  exit if $l \geq \delta^k$
  determine $\delta(Q,O)$
  $result$.insert($\delta(Q,O), oid(O)$)
  if $result$.getNumber()>$k$ then
    $\delta^k$ = $result$.getDist($k$)
until $heap$.isEmpty()
```

5. EXPERIMENTS

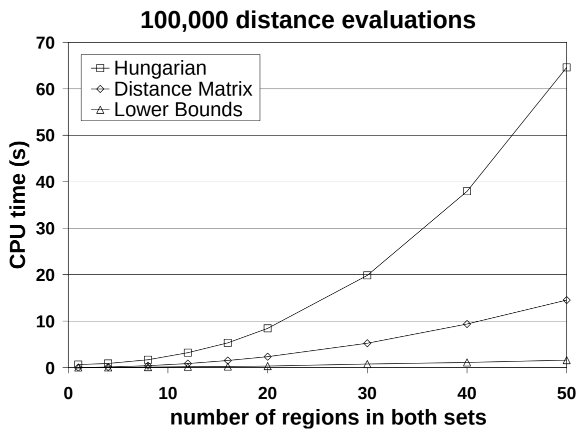
In the following, we present efficiency experiments of the proposed multi-step algorithms and show the tightness of the bounds. Note that these experiments do not contain any effectiveness results of the proposed retrieval model as its superiority over other models already was described in [2].

Data Set. Over the past years, we have established a large image retrieval system with around 370,000 images that is capable to answer CBIR and RBIR queries[3]. Out of this database, we used two feature sets: Color5x5 (300MB) and ColorJSEG (6MB). For the former one, we applied a static partitioning layout of 5x5 overlapping regions on all 370,000 images. For the latter data set, we applied the JSEG segmentation algorithm [7] on around 19,000 images resulting in 9 regions on average. In both data sets, each region was described by a color moment feature [18] with 9 dimensions. The distance between two regions was determined by a weighted Manhattan distance.

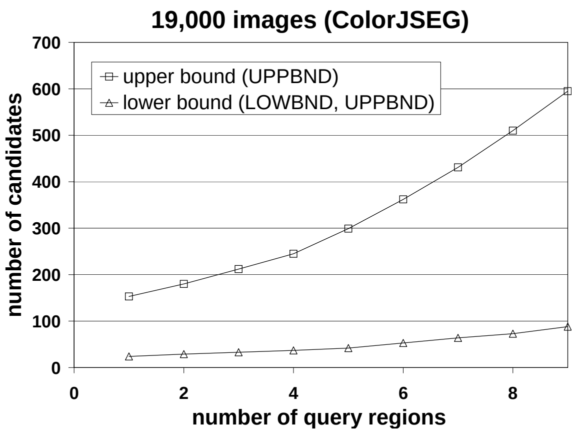
Platform. All measurements have been performed on an Intel Pentium 4 with 1.8 GHz, 256 MB main memory, and a hard disk with a throughput of 25 MB/s. The platform was running Microsoft Windows 2000 Server. For the experiments, we used our C++ implementation. In all settings, we always read data from the hard disk bypassing any caching in the operating system.

Queries. All experiments run *"contains"* queries with penalties $p_1 = 0$ and $p_2 = \infty$. The number of query regions varied between the experiments. Measurements with other query types revealed similar results. For ColorJSEG and

[3]The demo is available at `http://www-dbs.ethz.ch/imagedb`. Due to attacks, we had to close the new RBIR part.



(a)



(b)

Figure 2: (a) Costs for distance computations; (b) Lower and upper bound candidates.

Color5x5, we averaged the measurements over 100 queries of the same type. If not stated differently, we were searching for the 20 nearest neighbors.

5.1 Number of Candidates

The first experiment investigates the numbers of candidates for the proposed multi-step algorithms. Figure 2(b) illustrates typical results for the data set ColorJSEG. It depicts the number of lower and upper bound candidates as a function of the number of query regions. Obviously, lower bounds are very tight since less than 0.5% (95 out of 19,000) of the objects qualify as a candidate with *LOWBND* and *UPPBND*. The size of the upper bound candidate set of *UPPBND* is by a factor of around 6 larger than the one of the lower bound candidate set. In the worst case, this set contains about 3% of the database objects. Consequently, *UPPBND* is able to hold the distance matrices of candidates between the two steps since this cache requires only a small amount of main memory.

5.2 Retrieval Performance

Next, we evaluate the multi-step algorithms with queries over the larger data set Color5x5. Figure 3 compares the CPU costs (left hand side) and elapsed times (right hand side) of the algorithms by varying the number of query regions (each object in the database contained 25 regions). In Figure 3 (a), we observe that the CPU costs for *SCAN* are growing at a much faster rate than the ones of the multi-step algorithms: with 25 query regions, CPU costs are 5 times larger. Surprisingly, the multi-step algorithms have similar CPU costs. In other words, the heap maintenance costs (CPU) in *LOWBND* are negligible.

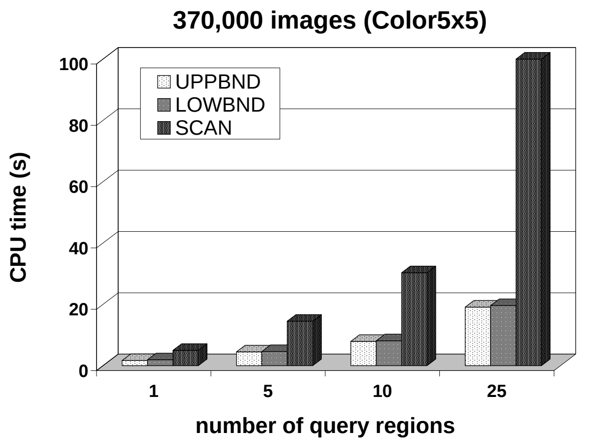
Figure 3 (b) depicts elapsed times for region-based searches. Due to large CPU costs, the performance of *SCAN* degenerates when we increase the number of query regions. Elapsed times for the multi-step algorithms are very similar if the number of query regions is smaller than 10. After this point, the performance of *LOWBND* becomes worse compared to the other multi-step approaches: with 25 query regions, *LOWBND* needs two times longer to find the best matches. This difference accrues from revisiting candidates in the second step. Thereby, objects are accessed in random order from the hard disk. With access costs of 10 ms and 2000 candidates, the second step suffers from additional IO costs in the order of 20 seconds. *UPPBND*, on the other hand, is able to cache candidate objects between the two steps and, hence, no additional IO costs accumulate. We also note that region-based retrieval becomes a CPU-bound retrieval problem, if the number of query regions exceeds around 15 regions in our setting. With less than 15 regions, the IO costs to read 300 MB from hard disks hinders faster response times.

Finally, we conducted experiments by varying the number k of results to return. Figure 4(a) depicts these results for 25 region queries over the data set Color5x5 (results with the smaller data set look similar). Over the entire range of k, the search costs for both the brute force search SCAN and the multi-step method *UPPBND* remain constant. Hence, retrieval costs with multi-step methods are stable over a large range of values for the number k of objects to retrieve.

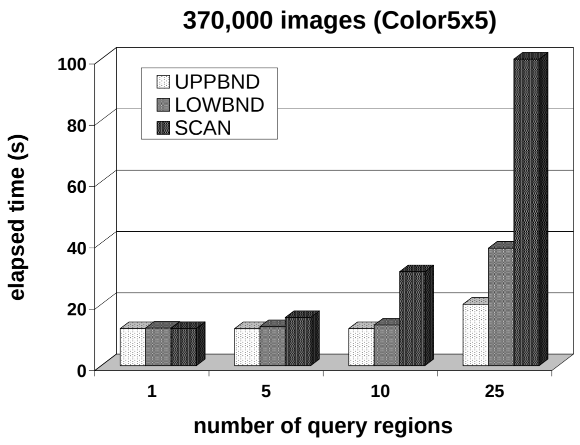
5.3 Comparisons With Approximate Methods

Next, we compare the accurate algorithms *UPPBND* and *SCAN* with the approximate methods of Blobworld and WALRUS. Recall that these systems issue a search for each query region in isolation, and combine the retrieved results to obtain an overall result. Furthermore, note that the retrieval models of Blobworld and WALRUS have a worse effectiveness compared to the model deployed in this paper and the Windsurf system [2]. We have implemented the RBIR model of Blobworld and WALRUS based on an extension of the VA-File [20] which can evaluate a batch of queries in parallel. To obtain a result of reasonable quality, we issued a 100 nearest neighbor search for each query region. Then, we combined the results according to the rules of Blobworld and WALRUS, and identified the 20 best images.

Figure 4(b) illustrates the elapsed times of the algorithms



(a)



(b)

Figure 3: Comparison of execution costs: (a) CPU times, and (b) elapsed times.

for the cases: data set ColorJSEG with 2 and 10 query regions, and data set Color5x5 with 2 and 25 query regions (note the two different y-scales). Surprisingly, we observe that the retrieval with Blobworld and WALRUS outperforms multi-step retrieval only in a few cases. Especially, if the number of query regions is large, Blobworld/WALRUS searches run twice as long as multi-step searches. This is because batch evaluation in multi-dimensional indexes with only a few query regions is concentrated to a small data partition in the index. If the batch contains more query regions, larger parts of the index must be considered. Thereby, more data is fetched randomly from the disk. Hence, while IO costs for multi-step retrieval remain constant when increasing the number of query regions, the ones for approximate retrieval grow linearly. We conclude that the more effective retrieval model of the Windsurf system does not lead to larger retrieval costs than the evaluation of the less effective models of Blowbworld and WALRUS.

6. CONCLUSIONS

Region-based image retrieval (RBIR) is an extension of the classical content-based image retrieval (CBIR). RBIR systems provide new query types, most importantly, to search for objects embedded in an arbitrary environment. For multi-region queries, we deployed an extension of the dissimilarity function of Windsurf [2] which is more intuitive and effective compared to alternative models. Windsurf formulates an Assignment Problem to find a matching between the reference regions and the image regions. The main problem of this approach is the expensive evaluation of distances between two region sets. Using a brute-force search, RBIR retrieval is limited to small size databases. In this work, we have described efficient multi-step algorithms to search for the k best matches for RBIR queries. Our bounding distance functions are so tight that less than 0.5% of the database remain after the first (filtering) step. Experiments have demonstrated that multi-step algorithm outperform a brute-force method by more than a factor of 5, and perform equally well or even better than less effective retrieval methods.

As future work, we attack the problem of region-based retrieval from several angles: first, we are searching for cheaper but still tight lower bounding functions. Especially, we are interested in functions which do not require the distance matrix between the two region sets. Second, we like to extend our retrieval model to cover spatial constraints. However, such a model is only reasonable if there exists an efficient retrieval algorithm or indexing structure. Third, we want to further decrease response times for region-based queries. Since our multi-step algorithm have a linear structure, they are ideal for grid computing and an almost linear speedup of retrieval should be feasible.

7. REFERENCES

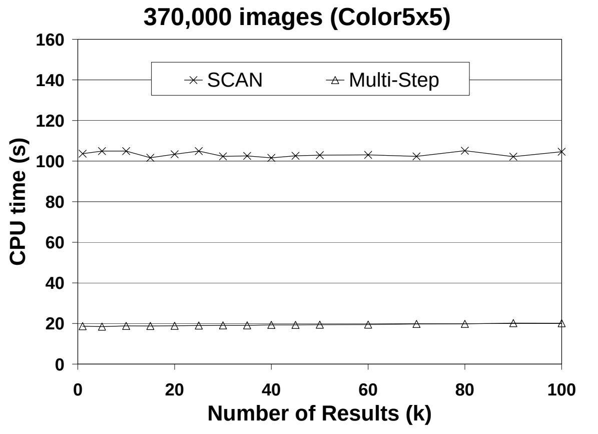
[1] M. Ankerst, B. Braunmüller, H.-P. Kriegel, and T. Seidl. Improving Adaptable Similarity Query Processing by Using Approximations. In *Proc. of the Int. Conference on Very Large Databases*, p. 206–217, New York City, NY, USA, 1998.

[2] I. Bartolini, P. Ciaccia, and M. Patella. A Sound Algorithm for Region-Based Image Retrieval Using an Index. In *Proceedings of the 4th International Workshop on Query Processing and Multimedia Issues in Distributed Systems (QPMIDS'00 - DEXA'00)*, p. 930–934, Greenwich, London, U.K., 2000.

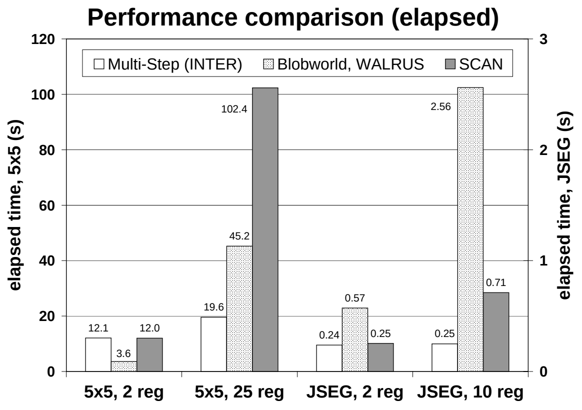
[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-Tree: An Index Structure for High-Dimensional Data. In *Proc. of the Int. Conference on Very Large Databases*, p. 28–39, Mumbai (Bombay), India, 1996.

[4] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "Nearest Neighbour" Meaningful? In *Proc. of the Int. Conf. on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, p. 217–235, Jerusalem, Israel, 1999.

[5] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A System for Region-Based



(a)



(b)

Figure 4: (a) Influence of the number of results; (b) Comparison of correct and approximate retrieval.

Image Indexing and Retrieval. In *3rd International Conference on Visual Information and Information Systems*, volume 1614 of *Lecture Notes in Computer Science*, p. 509–516, Amsterdam, The Netherlands, 1999.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of the Int. Conference on Very Large Databases*, p. 426–435, Athens, Greece, 1997.

[7] Y. Deng and B. S. Manjunath. Unsupervised Segmentation of Color-Texture Regions in Images and Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):800–810, 2001.

[8] C. Faloutsos. Access Methods for Text. *ACM Computing Surveys*, 17(1):49–74, 1985.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, p. 419–429, Minneapolis, Minnesota, United States, 1994.

[10] V. Gouet and N. Boujemaa. Object-based queries using color points of interest. In *IEEE Workshop on Content-Based Access of Image and Video Libraries (CBAIVL 2001)*, p. 30–36, Kauai, Hawaii, USA, 2001.

[11] J. Hebborn. *Decision Maths 2*, chapter 2. Allocation (assignment) problems, p. 29–45. Heinemann Modular Maths for Edexcel AS and A-level. Heinemann, 2001.

[12] L. Hermes, T. Zöller, and J. M. Buhmann. Parametric Distributional Clustering for Image Segmentation. In *Proceedings of the 7th European Conference on Computer Vision (Part 3)*, volume 2352 of *Lecture Notes in Computer Science*, p. 577–591, Copenhagen, Denmark, 2002.

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

[14] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast Nearest Neighbor Search in Medical Image Databases. In *Proc. of the Int. Conference on Very Large Databases*, p. 215–226, 1996.

[15] A. Natsev, R. Rastogi, and K. Shim. WALRUS: a similarity retrieval algorithm for image databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, p. 395–406, Philadelphia, PA, USA, 1999.

[16] Y. Rui, T. S. Huang, and S.-F. Chang. Image Retrieval: Current Techniques, Promising Directions and Open Issues. *Journal of Visual Communication and Image Representation*, 10(4):39–62, 1999.

[17] T. Seidl and H.-P. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Seattle, Washington, 1998.

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

[19] J. Z. Wang. SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 23(9), p. 947–963, 2001.

[20] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. of the Int. Conference on Very Large Databases*, p. 194–205, New York City, New York, USA, 1998.