University of Alberta

# Color Based Image Retrieval Using Signature Trees

by

Mario A. Nascimento
Eleni Tousidou
Vishal Chitkara
Yannis Manolopoulos

Technical Report TR 01-02
March 2001

DEPARTMENT OF COMPUTING SCIENCE
University of Alberta
Edmonton, Alberta, Canada

# Color Based Image Retrieval Using Signature Trees

Mario A. Nascimento
mn@cs.ualberta.ca
Vishal Chitkara
chitkara@cs.ualberta.ca

Dept. of Computing Science
Univ. of Alberta, Edmonton, Canada

Eleni Tousidou
etousido@auth.gr
Yannis Manolopoulos
manolopo@auth.gr

Dept. of Informatics
Aristotle Univ., Thessaloniki, Greece

March 20, 2001

**Abstract**

Significant research has focused on determining efficient methodologies for effective and speedy retrieval in large image databases. Towards that goal, the first contribution of this paper is an image abstraction technique, called VBA, based on signature bit-strings and a corresponding similarity metric. The signature provides a compact representation of an image based on its color content and yields better retrieval effectiveness than when using classical global color histograms (GCHs) and comparable to the one obtained when using Color-Coherence Vectors (CCVs). More importantly however, the use of VBA signatures allows savings of 75% and 87.5% in storage overhead when compared to GCHs and CCVs, respectively. The second contribution is the improvement upon an access structure, the S-tree, exploring the concept of logical and physical pages and a new nearest-neighbor type of algorithm, in order to improve retrieval speed. We compared the S-tree performance when indexing the VBA signatures against the SR-tree indexing GCHs and CCVs, since SR-trees are arguably the most efficient access method for high-dimensional points. Our experimental results, using a large number of images and varying several parameters, have shown that the combination VBA/S-tree outperforms the GCH/SR-tree combination in terms of effectiveness, access speed and size (up to 45%, 25% and 70% respectively). Due to the very high-dimensionality of the CCVs their indexing, even using an efficient access structure, the SR-tree, did not seem to be a feasible alternative.

## 1 Introduction

The enormous growth of image archives has significantly increased the demand for research efforts aimed at efficiently finding similar images within a large image database (e.g., [9, 24]). One popular strategy of searching for images within an image database is called Query By Example (QBE), in which the query is expressed as an image template or a sketch thereof, and is commonly used to pose queries in most Content-Based Image Retrieval (CBIR) systems, for instance, IBM's QBIC [16], Virage's VIR [34], WebSEEk [27] and SIMPLIcity [35].

Typically, the CBIR system extracts visual features from a given query image, which are then used for comparison with the features of other images stored in the database. The similarity function is thus based on the abstracted image content rather than the image itself. The color distribution of an image is one such feature that is extensively utilized to compute the abstracted image content. It exhibits desired properties, such as low complexity for extraction, invariance to scaling and rotation, and partial occlusion [30]. In fact, it is common to use a *Global Color Histogram* (GCH) to represent the distribution of colors within an image. Assuming an $n$-color model, a GCH is an $n$-dimensional feature vector $(h_1, h_2, ..., h_n)$, where $h_j$ represents the (usually) normalized percentage of color pixels in an image corresponding to each color $c_j$. In this context, the retrieval of similar images is based on the similarity between their respective GCHs. A common similarity metric is based on the Euclidean distance (though other distances could also be used) between the abstracted feature vectors that represent two images, and it is defined as: $d(Q,I) = \sqrt{\sum_{j=1}^{n}(h_j^Q - h_j^I)^2}$ where $Q$ and $I$ represent the query image and one of the images in the image set, and $h_j^Q$ and $h_j^I$ represent the coordinate values of the feature vectors of these images respectively. Other approaches to abstract an image for the sake of indexing and searching will be reviewed in Section 2.

It should be noted though that, when using the GCH approach, storing $n$-dimensional vectors of a color histogram for each image in the database may consume significant storage space. In order to minimize the space requirements, we propose the use of a compact representation of these vectors, thereby leading us to the utilization of binary signatures. An image's signature bit-string, hereafter referred to as *signature*, is an abstract representation of the color distribution of an image by bit-strings of a pre-determined size. However, it is important to note that careful use of signatures yields not only much smaller space overhead, but also considerably higher retrieval effectiveness. This will be discussed in Section 3.

Further to the above observation, there is the problem of efficient, i.e., fast, searching similar images in a large database. Mapping color histograms onto points in an $n$-dimensional space is an ellegant way to solve the problem, but it has a serious drawback. It makes the problem of searching similar images using a disk-based access structure a much harder problem as the value of $n$ grows. In fact, it is not unusual to use values of $n$ in excess of 64. This renders the use of traditional spatial access structures (e.g., the R$^*$-tree [2]) of little use. To address this drawback we propose the use of an enhanced Signature Tree (S-tree) [10, 32]. As we will present in Section 4, the improved S-tree can take advantage of the concept of logical and physical pages. In addition, we propose new algorithms for fast nearest-neighbor queries for the S-tree.

In Section 5 we investigate thoroughly the performance of the proposed approach in terms of both effectiveness and efficiency. The obtained results indicate the combination VBA/S-tree to be robust, quite effective and very efficient.

Summarizing, the main contributions of this paper are: the proposal of a new and compact representation of images based on their color distribution; the enhancement upon the S-tree to process nearest-neighbors queries more efficiently; the empirical demonstration that the binary signatures are an effective way to represent an image and that the combination of image signatures and the S-tree is a sound, effective and efficient solution for the problem of color-based image retrieval.

## 2 Related Work

In recent years, there has been considerable research published regarding CBIR systems and techniques. In what follows, we give an overview of some representative work related to ours, i.e., color-oriented CBIR.

QBIC [16, 12] is a classical example of a CBIR system. It performs CBIR using several perceptual features, e.g., colors and spatial-relationships. The system utilizes a partition-based approach for representing color. The retrieval using color is based on the average Munsell color and the five most dominant colors for each of these partitions, i.e., both the global and local color histograms are analyzed for image retrieval [20]. Since the quadratic measure of color distances is computationally intensive, the average Munsell color is used to pre-filter the candidate images. The system also defines a metric for color similarity based on the bins in the color histograms.

A similarity measure based on color moments is proposed in [29]. The authors propose a color representation that is characterized by the first three color moments namely color average, variance and skewness, thus yielding very low space overhead. Each of these moments is part of an index structure and has the same units, which make them somewhat comparable to each other. The similarity function used for retrieval is based on the weighted sum of the absolute difference between corresponding moments of the query image and the images within the data set.

A similar approach was also proposed by Appas et al [1], the main difference being that the image is segmented in five overlapping cells. The use of a superimposed $4\times4$ non-overlapping grid has been proposed in [25].

Another technique for integrating color information with spatial knowledge in order to obtain an overall impression of the image is discussed in [15]. The technique is based on the following steps. By using a similar grid of cells, the authors propose some heuristics to capture relevant colors and distinguish among background and non-background (main object) colors.

A system for color indexing based on automated extraction of local regions is presented in [26]. The system first defines a quantized selection of colors to be indexed. Next, a binary color set for a region is constructed based on whether the color is present or not in a region. In order to be captured in the index by its color set, a region must meet the following two requirements: (i) there must be at least $N$ pixels in a region, where $N$ is a user-defined parameter, and (ii) each color in the region must contribute with at least a certain percentage of the total region area; this percentage is also user-defined. Each region in the image is represented using a bounding box. The information stored for each region includes the color set, the image identifier, the region location and the size. The image is therefore queried, based not only on the color, but also on the spatial relationship and composition of the color region.

In [28] the authors attempt to capture the spatial arrangement of different colors in the image, based on using a grid of cells over the image and a variable number of histograms, depending on the number of distinct colors present. The paper argues that on the average, an image can be represented by a relatively low number of colors, and therefore some space can be saved when storing the histograms. The similarity function used for retrieval is based on a weighted sum of the distance between the obtained histograms. The experimental results have shown that such a technique yields 55% less space overhead than conventional partition-based approaches, while still being up to 38% more efficient in terms of image retrieval.

Pass et al [22] describe a technique based on incorporating spatial information with the color histogram using Color-Coherence Vectors (CCVs). The technique classifies each pixel in a color bucket as either coherent or incoherent, depending upon whether the pixel is a constituent of a large similarly-colored region. The argument is that the comparison of coherent and incoherent feature vectors between two images allows for a much finer distinction of similarity than when using color histograms. Note that using CCVs one will have two histograms for each image (one for coherent colors and one for incoherent colors); each one as large as traditional GCH. The authors compare their experimental results with various other techniques and show their technique to yield a significant improvement in retrieval performance.

Recent proposals include [19] and [35]. Lin proposes a method based on multi-precision similarity. The idea is to recursively partition the image in a number of non-overlapping cells, for which the average RBG vector is recorded. This recursive tiling takes into account spatial distribution of colors as well as it allows querying sub-images. However, there is no strong evidence whether that method actually improves retrieval effectiveness. Finally, Wang's SIMPLIcity uses a wavelet-based approach to extract features, being able to segment images in real-time. Among the many possible usable features to measure similarity, the system uses a "global" region-matching scheme, which is claimed to be robust to poor segmentation. The CBIR approach underlying SIMPLIcity can also perform image categorization, a task which is usually not carried out.

We conclude this section with a brief survey about how CBIR research has used access structures. This is an important aspect as the number of images in a database may render any retrieval approach based on linear scanning of the image (metadata) file unfeasible.

As discussed earlier, images are mapped to a high-dimensional feature space. Most of the time, this feature space can be mapped into the Euclidean space and spatial access structures [13] can be used. Although typical spatial access structures are not well suited for high-dimensions (e.g., R$^*$-trees [2]), recent proposals have been made for that case, e.g., the X-tree [3], SS-tree [36] and the SR-tree [18] to name a few. The X-tree makes use of the concept of super-node in order to minimize overlap between the area covered by different sub-trees; it was shown to be more efficient than the R$^*$-tree. Unlike the R$^*$-tree, which uses minimum bounding boxes to represent the area covered by a sub-tree, the SS-tree uses minimum bounding spheres and was shown to outperform the R*-tree. We are not aware of any research comparing the X-tree to the SS-tree directly. The SR-tree capitalizes on the fact that one can use both boxes *and* spheres in the internal nodes in order to decrease overlap between the area covered by the sub-trees. It was reported to be more efficient than both the R*-tree and the SS-tree, and has lately been regarded as the best access structure for high-dimensional data.

For some cases though, it is not trivial (or possible) to map the feature space into the Euclidean space. For those situations the use of metric spaces is usually a good (or perhaps the only) solution. Recent works on access structures for indexing metric spaces are the M-tree [6] and the Slim-tree [33]. A key issue for an efficient metric tree is an accurate and computationally non-complex distance metric. Unfortunately, it is often the case that an accurate metric is complex. This, in turn, may render those structures CPU-bound instead of I/O-bound, as usual in this domain.

A comparative study with respect to efficiency (i.e., storage overhead and query processing time) using the R*-tree, SS-tree, SR-tree and the M-tree was carried out in [7]. The main conclusions were that: (i) the SR-tree is overall the most efficient of the four access structures being able to cope better with the increase of the space dimensionality; (ii) the SR-tree is also able to take advantage of larger nodes; (iii) the SS-tree is the most compact one and (iv) I/O counts may indeed be a misleading indication of performance, specially in the case of the M-tree where complex metrics can turn the process CPU-bound.

It is also interesting to note that among the CBIR approaches reviewed above, few, [12] being one such example, make use of disk-based access structures to speedup query processing. In this paper we propose not only a CBIR method (next Section) but also an efficient disk-based structure, the S-tree (Section 4) and an algorithm to support efficiently similarity queries.

# 3 VBA – A Scheme for Compact Image Signatures

The effectiveness of a CBIR system ultimately depends on its ability to accurately identify the relevant images. Our basic motivation is based on the observation that classical techniques based on GCHs often offer poor performance since they treat all colors equally and take only their distributions into account. More specifically, the relative density of a color is not taken into account by these approaches. Let us motivate this point through a simple example. Consider two different pictures of a single colorful fish with two quite different and large backgrounds. The similarities/differences among the many less dense colors of the fish are likely more important than the large differences in the background. Hence, we believe that colors relatively less dense should be given more attention.

The majority of the efforts reviewed in Section 2 have been directed towards a representation of those colors in the color histogram that have a significant pixel dominance. Our approach differs mainly in this regard. We actually emphasize more on the colors which are less dominant while still taking major colors into account. In order to use signatures for image abstraction, we have designed the following scheme:

- Each image in the database is quantized into a fixed number of $n$ colors, $C = (c_1, c_2, \ldots, c_n)$ to eliminate the effect of small variations within images and also to avoid using a large file due to the high resolution representation [21].

- Each color element $c_j$ is then discretized into $t$ binary bins $(B^j = b_1^j b_2^j \ldots b_t^j)$. Each bin is characterized by a *capacity*, referred to as the *bin-size*, which represents the range (in percentage) of dominance of the $j^{th}$ color in the image (e.g. the $j^{th}$ color appears in the image with a frequency within a certain percentage range). As an example, consider an image comprising of $n$ colors and $t$ bins per color. The signature of this image would then be represented by the bit-string: $S = b_1^1 b_2^1 \ldots b_t^1 b_1^2 b_2^2 \ldots b_t^2 \ldots b_1^n b_2^n \ldots b_t^n$ where, $b_i^j$ represents the $i^{th}$ bin relative to the color element $c_j$. For simplicity, we refer to the sub-string $b_1^j b_2^j \ldots b_t^j$ as $B^j$ $(1 \leq j \leq n)$, hence, the signature of an image $I$ can also be denoted as: $S_I = B_I^1 B_I^2 \ldots B_I^n$.

The key point then becomes how to set the bits $b_i^j$. Bin-sizes could be equal or varying, i.e. they could represent equal or different range lengths, respectively. Our first attempt [4] to accomplish that was by using bins of equal capacities. In such a case each color $c_j$ has its bins set according to the following condition:

$$b_i^j = \begin{cases} 1 & \text{if } i = \lceil h_j \times t \rceil \\ 0 & \text{otherwise} \end{cases}$$

where $h_j$ is an element of the feature vector that represents the GCH as discussed in Section 1.

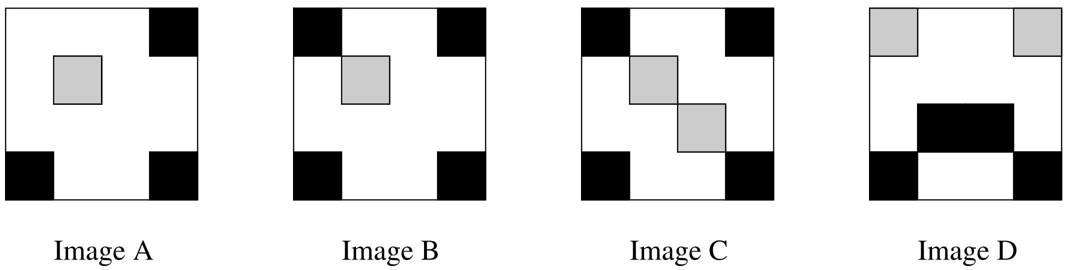


Figure 1: Sample image set

As an example, consider image A in Figure 1 having three colors, and for simplicity, let us assume $n$=3, hence, $C = (c_1, c_2, c_3)$ = (black, grey, white). The normalized color densities can then be represented by the vector $(h_1, h_2, h_3)$ = (0.18, 0.06, 0.76), where each $h_j$ represents the percentage pixel dominance (relative ratio) of color $c_j$. Next, assume that the color distribution is discretized into $t$=10 bins of equal capacities. Hence, $b_1$ would accommodate color dominance (pixel-wise) from 1% to 10%, and $b_2$ would accommodate from 11% to 20% and so on. Therefore, image A can then be represented by the following signature $S_A$ = 0100000000 1000000000 0000000100.

However, we have also observed that less dominant colors comprise a significant part of an image. In fact, in our previous experiments [4], we verified that each image has, on average, about 11 colors. In addition, most of such colors have a density of 10% or less. Hence, it is conceivable to conjecture that a substantial cover of an image is due to many colors which individually constitute a small portion of the whole image. This led us to modify the signatures in order to emphasize colors with smaller densities by developing the VBA[1] scheme, which is presented next.

*Variable-Bin Allocation* (VBA) is based on varying the capacities of the bins representing the color densities. The design of VBA is based on the fact that distances due to less dominant colors are important for an efficient retrieval. The generated signature therefore lays a greater emphasis on the distance between the less dominant colors than on more dominant ones. The experiments performed on VBA, assuming a color distribution discretized into $t = 10$ bins, were based on $b_1$ and $b_2$ having bin-sizes equal to 3, $b_3$ having bin-size equal to 4, $b_4$ and $b_5$ having bin-sizes equal to 5, bins $b_6$ through $b_9$ having bin-sizes equal to 10, and, finally, bin $b_{10}$ having the bin-size equal to 40. For instance, if a color covers between 4% and 6% of an image the bit corresponding to $b_2$ is set and all other bits remain unset. Note that, while on the one hand there is a fine bin-wise granularity for colors covering a small portion of the image (and most do, as argued above), a color covering between 61% and 100% of an image will, on the other hand, set a single bit as well. This is due to the fact that an image rarely has more than half of itself covered by a single color. In other words, we enhance the granularity of colors with smaller distributions, without completely ignoring those with a large presence. For the sake of illustration, the VBA signatures for all images in Figure 1 are shown in Table 1.

[1]This VBA scheme presented in this paper is a slight modification of the one presented originally in [4]. However, one must note that in that paper the issue of efficient indexing of signature using secondary memory was not touched upon.

| Color/ Set of Bins | Color Density | Binary Color Signatures |
|---|---|---|
| $c_1/B_A^1$ | 18% | 0 0 0 0 1 0 0 0 0 0 |
| $c_2/B_A^2$ | 6% | 0 1 0 0 0 0 0 0 0 0 |
| $c_3/B_A^3$ | 76% | 0 0 0 0 0 0 0 0 0 1 |
| $c_1/B_B^1$ | 24% | 0 0 0 0 0 1 0 0 0 0 |
| $c_2/B_B^2$ | 6% | 0 1 0 0 0 0 0 0 0 0 |
| $c_3/B_B^3$ | 70% | 0 0 0 0 0 0 0 0 0 1 |
| $c_1/B_C^1$ | 24% | 0 0 0 0 0 1 0 0 0 0 |
| $c_2/B_C^2$ | 12% | 0 0 0 1 0 0 0 0 0 0 |
| $c_3/B_C^3$ | 64% | 0 0 0 0 0 0 0 0 0 1 |
| $c_1/B_D^1$ | 24% | 0 0 0 0 0 1 0 0 0 0 |
| $c_2/B_D^2$ | 12% | 0 0 0 1 0 0 0 0 0 0 |
| $c_3/B_D^3$ | 64% | 0 0 0 0 0 0 0 0 0 1 |

Table 1: Detailed Signatures of the images in Figure 1 using VBA

Recall that each bin is represented by a single bit. Hence, the obtained signature is a compact, yet effective, representation of the color content of an image. In addition, as we shall see later in this paper, this representation allows effective retrieval as well.

It is clear that at most a single bit is set within each set $B_j$ of bins. Hence, one could simply record the position of this single bit within $B_j$ which is set, instead of recording the whole 10 bit signature for $B^j$. If each set $B^j$ of bins comprises of $t$ bins, only $\lceil \log_2 t \rceil$ bits are needed to encode the position of the set bit (if any). Thus, for $t = 10$, the VBA approach would require a mere 4 bits to encode each color. As a comparison, let us assume that the storage of a real number requires $f$ bytes. Storing a GCH of an image comprising of $n$ colors, would require $(n \times f)$ bytes. Indeed, by using $n = 64$, $f = 2$ and $t = 10$, the VBA approach would require only 32 bytes compared to the 128 bytes required by the GCH, a substantial savings of 75% in storage space.

The savings are even more respectable when compared to other well known approaches, e.g., the CCV presented in [22]. Incidentally, CCVs are one of the approaches against which we will compare our proposed technique later in the paper. A CCV is in fact a pair of global histograms, one for coherent colors, and another one for incoherent colors. Therefore it is twice as large as a GCH. This means that by using a VBA signature we obtain 87.5% savings over the space required by CCVs.

If we assume that an image's address in disk consumes 8 bytes, then using 4 Mbytes of main memory one would be able to "index" *and* search a mid-size image database of roughly 100,000 images. After an initial linear scan on the signature file to load the signatures onto memory, all queries could be processed I/O free. Perhaps, more importantly, one would be able to do so avoiding the overhead of maintaining a disk-based access structure. This is an important result as it allows to implement easily speedy searches for images in many application domains, e.g., digital libraries. However, such an approach would not scale for very large image databases, say millions of images. For that purpose we propose the use of the S-tree, which we discuss in details in the next section.

Once the signatures for each image in the data set are pre-computed and stored into the database, the retrieval procedure can take place. To assess the similarity between two images, $Q$ and $I$, we will use a metric akin to the $L_2$ Euclidean distance, defined as follows:

$$d(Q, I) = \sum_{j=1}^{n} \left[ pos(B_Q^j) - pos(B_I^j) \right]^2$$

where $pos(B_R^k)$ gives the position of the set bit within the set $B^k$ of bins for image $R$. For instance, using image A in Figure 1 and its VBA signature, we have $pos(B_A^1) = 5$, $pos(B_A^2) = 2$ and $pos(B_A^3) = 10$. We have shown in [4] that this metric is more robust than its $L_1$ counterpart.

At this point it is worthwhile to single out the most important disadvantage of using only the color distribution as a basis to abstract an image. It is not unreasonable to argue that images C and D (Figure 1) are not similar. Even though they do have the same color distributions, the colors have very different *spatial* distribution, which is *not* captured when only the quantitative color density is used, in fact, $d(C, D) = 0$. However, this is not a problem due to our abstraction scheme nor to our metric, but due to the use of only color distributions, e.g., GCHs. We must also note that we have introduced new variables to the problem of image abstraction. Whereas one would only have to choose the number of colors $n$ into which the original images should be quantized, now the new variable $t$ also exists.

# 4 S-tree – An Access Structure for Binary Signatures

So far, the most common use of these binary bitstrings, or else signatures, was to index text or to indicate the presence or absence of individuals in sets [17, 14]. For example, when used in object-oriented databases they would represent the existence of the different items that constitute the set-valued attribute of an object. Each element of a specific set was encoded by using a hashing function into a signature of length $F$ and *weight*, i.e., number of set bits, $m$. The object's signature was generated by applying the superimposed coding technique on all element signatures, i.e. all positions were superimposed by a bit-wise OR-operation to generate the set's signature.

As described in the previous section, in the context of image retrieval, signatures can also be used to represent the frequency of a specific color appearing in a given image. Each color uses $t$ binary bins (i.e., positions), where each bin depicts a range of frequencies for the specific color. A position in the signature is set only when the represented color exists in the image within the corresponding frequency range. However, unlike to the case of set-valued attributes, image signatures will be constructed by concatenating, and not superimposing, their color signatures.

A collection of signatures comprises a signature file, which presents low space overhead and reduced update costs [5, 11]. Trying to improve the performance of signature files and avoid the sequential scanning that they introduce, a number of tree- and hash-based signature organizations have been designed. Such an approach is the S-tree which, similarly to a B$^+$-tree [8], is a height balanced dynamic structure [10].

Each node of an S-tree contains a number of pairs $< s, p >$, where $s$ is a signature and $p$ is a pointer to a child node. The S-tree is defined by two integer parameters: $M$ and $m$. The root can accommodate at least 2 and at most $M$ pairs, whereas all other nodes can accommodate at least $m$ and at most $M$ pairs. Unlike B$^+$-trees where $m = M/2$, here it holds that: $1 \leq m \leq M/2$. Thus, the tree height for $n$ signatures is at most: $h = \lceil \log_m n - 1 \rceil$. Signatures in internal nodes are formed by superimposing the signatures of their children nodes. In our context, the leaves of the S-tree will contain image signatures, along with a unique identifier for those images. In the following we will show how the S-tree can be used as an efficient and effective tool to store and search the aforementioned image signatures.

## 4.1 Using the S-tree in Image Retrieval

During the insertion of a new object in an S-tree, the leaf in which its signature will be stored is selected by traversing the tree in a top-down order and by choosing the most appropriate node at each level according to a distance criterion. When the appropriate leaf is reached, it is possible that it is already full, i.e., it contains $M$ entries, and therefore it will have to be split. A new node will be created and the $M$+1 entries will be distributed between the two nodes, also in an appropriate way. When inserting an object or when splitting a node, the aim is to cluster the stored objects so that a well defined distance metric is minimized. As a consequence, the number of paths that will have to be traversed during searches is minimized as well.

The original S-tree [10] tried to minimize the node weight during insertions or node splits in order to render the inclusion query more efficient. However, for the current application, when deciding to store image signatures, the weight alone is not sufficient. Due to the way that the similarity distance is defined, the focus should be on the specific positions of the bits set in a signature and not only on their quantity.

Therefore, it is reasonable to expect that the dominant factor is the similarity distance: the smaller the distance between two images, the more similar are the images themselves. After experimenting (a) with the original distance function measuring the weight increase, and (b) with the function $d$ (defined in Section 3), it was obvious that the efficiency of the similarity search was drastically improved when the latter was applied.

For example (Table 2), if image $A$ were to be stored either with image $B$ or $C$, the weight increase equally in both cases, namely by 1 unit. On the other hand, the distance function $d$ would yield different results, 1 in the first case and 4 in the second. This is due to the fact that that image $A$ is more similar to image $B$ than to image $C$, hence it should be clustered with the former and not the latter.

| Image | Signatures | | |
|---|---|---|---|
| $A$ | 0000100000 | 0100000000 | 0000000001 |
| $B$ | 0000010000 | 0100000000 | 0000000001 |
| $C$ | 0000010000 | 0001000000 | 0000000001 |
| $(A \wedge B)$ | 0000110000 | 0100000000 | 0000000001 |
| $(A \wedge C)$ | 0000110000 | 0101000000 | 0000000001 |
| $d(A, B)$ | $= (6-5)^2 + (2-2)^2 + (10-10)^2 = 1$ | | |
| $d(A, C)$ | $= (6-5)^2 + (4-2)^2 + (10-10)^2 = 5$ | | |

Table 2: Calculation of function (image distance) $d$

### 4.2 Quadratic Split and Logical Pages

Studying the performance of the S-tree [10, 32, 31] it has been observed that due to the superimposition technique along the tree levels, nodes near the root tend to contain heavy signatures (i.e., with many 1s) and, thus, they present low selectivity. In [31, 32] several methods for the S-tree construction were introduced in order to address the problem and improve its search efficiency.

The original splitting algorithm of linear complexity can be viewed as consisting of two phases, the *seed selection phase* and the *signature distribution phase* [10]. During the first phase, the two most distant signatures with respect to a given distance are selected to play the role of seeds upon which the two new nodes will grow. When the seeds are chosen, the rest of the signatures are distributed to the two nodes according again to the same distance function.

Trying to reduce the low selectivity problem, a number of improved split methods of quadratic and cubic complexity are introduced in [32] which perform up to 5-10 times better when used in partial match queries. In this paper we adopt one of these methods, namely the *Quadratic Split*, since it offers good balance between an efficient search performance and node split (i.e., update) complexity. In particular, after choosing the two seeds as the most distant ones with respect to function $d$, we search for the entry with the maximum difference of the same function in the two nodes and insert it in the closer one. Here, it should be mentioned that only when a split occurs at the leaf level we make use of the actual distance $d$. In the upper levels, due to the superimposition, the actual distance cannot be applied and it is therefore substituted by the $minDist$ function, a similar function which will be described in more detail in the next section.

Moreover, based on results obtained in [31], a different structure for the tree leaves is also adopted. In the latter work, the performance of various structures was tested where the superimposed signatures stored above the leaf level were produced by a smaller number of signatures and not by the total number that exist in a leaf. The aim was to decrease the weight of the signatures that are stored in internal nodes and, consequently, increase their selectivity. The best approach was based on the use of logical pages, i.e. a number of independent logical pages were stored into a physical one. The number of logical pages per physical one is a parameter that can be tuned in order to achieve the required performance and trade off between space overhead and retrieval cost.

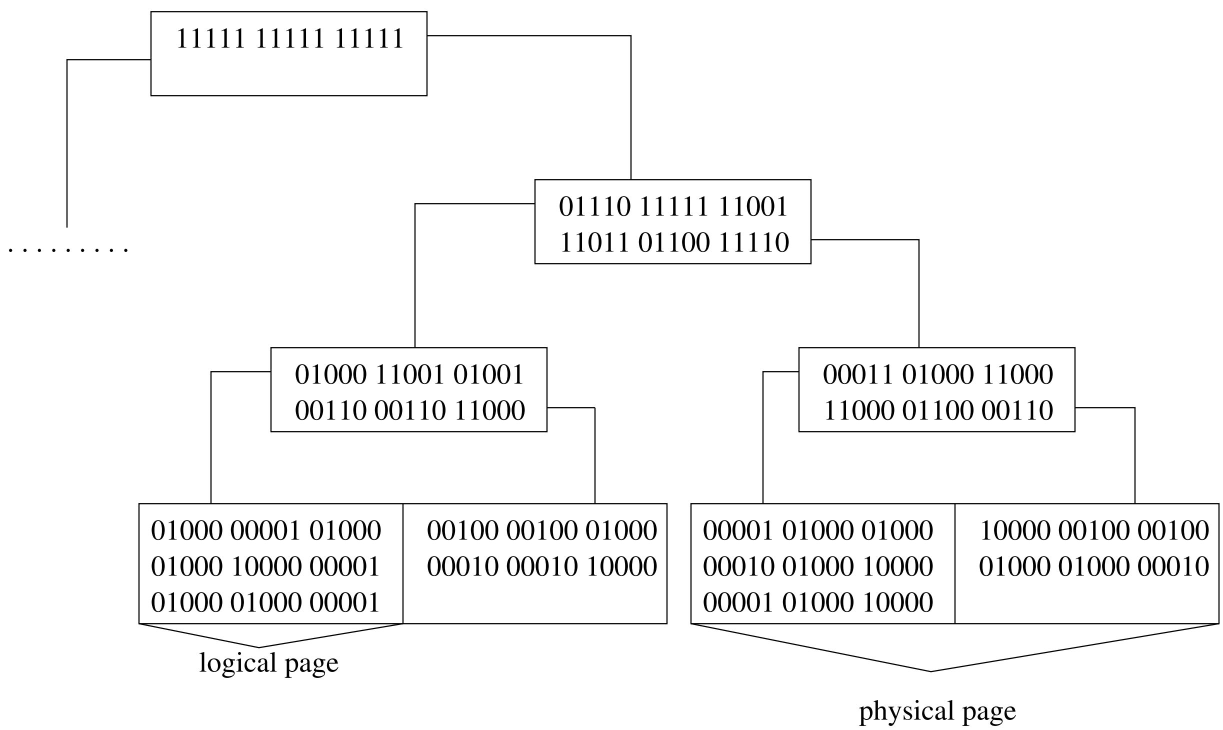


Figure 2: An S-tree containing logical pages at the leaf level.

For example, in Figure 2, image signatures will keep being inserted in an empty physical page until the first logical page is filled. If yet another signature is to be inserted in the same logical page, an overflow will occur. Then, following the Quadratic Split method the logical page will be split and one more logical page will be created. As illustrated in this example, each logical page produces its own superimposed signature which is stored in its parent node, ensuring this way its independence from the physical page.

### 4.3 Nearest-Neighbor Searching in S-trees

A popular query when data is represented by signatures is the partial match query, i.e., a subset or superset query that searches for all objects containing certain attributes. In such a case, the S-tree is traversed as follows. We descend the tree down to the leaf level following all paths where the stored signature evaluates the containment predicate with regard to the query signature. At the leaf level, all signatures satisfying the user query lead to the desired objects after discarding false-drops. In case of an unsuccessful path, searching may stop early at some level above the leaf level. However, in the present context the interest is in the nearest neighbor, or else, similarity query, i.e. to retrieve the $k$ most similar images to a given one.

In order to execute efficiently a similarity query in an S-tree, we have to embed drastic changes not only in the insert but also in the search procedure. More specifically, the methodology described in [23] is adopted, as a generalization of the nearest neighbor query to the $k$-nearest neighbors. This time, the adopted similarity function is function $d$ defined earlier.

While descending the tree though, a minimum distance also needs to be calculated in the internal nodes in order to make a first estimation of the images that are stored in the respective subtree. In particular, this second distance, called $minDist$, is related to an optimistic distance that can be calculated given a subtree signature, i.e., a signature in an internal node. The goal is to estimate which subtree can have an image closest to the query image. The procedure works as follows. For each color, we choose the bit which is physically closer to the bit set in the query image. The subtree signature which yields the minimum sum of distances is chosen as the best candidate.

Therefore, $minDist$ can be defined as

$$d(Q, I) = \sum_{j=1}^{n} \left[pos(B_Q^j) - optPos(B_I^j)\right]^2$$

where $optPos(B_R^k)$ gives the position of the physically closer set bit within the set $B^k$ of bins of the superimposed signature).

| | Color 1 | Color 2 | Color 3 |
|---|---|---|---|
| S | 0000110000 | 0100000000 | 0000000001 |
| S' | 0000010000 | 0001000000 | 0000000001 |
| Q | 0001000000 | 0000010000 | 0000000001 |
| $minDist$ | $(4-5)^2 + (6-2)^2 + (10-10)^2 = 17$ | | |
| $minDist'$ | $(4-6)^2 + (6-4)^2 + (10-10)^2 = 8$ | | |

Table 3: Calculation of $minDist$ function.

The example in Table 3, shows how the *minDist* distance is calculated. For example, for the first color, we consider that the closest image, with respect to that color, has an '1' in the fifth bin and not in the sixth one and so forth. Thus, if we were interested in finding out the $minDist$ of query $Q$ to the node $S$ containing the signatures for images $A$ and $B$, the respective $minDist$ of signature $S$ from query $Q$ would be 17. However, if we were interested in finding out the $minDist$ of query $Q$ from the node $S'$ containing the signatures of images $C$ and $D$, the respective $minDist$ of signature $S'$ from query $Q$ would be 8.

In the case of the single nearest neighbor query, when examining closer the search procedure, the tree is searched in a depth first search (similar results are obtained if a breadth first search is adopted). Whenever an internal node is reached, we compute the optimal distance $minDist$, i.e., the smallest possible distance that seems to exist in the respective subtree, and all children belonging to this node are inserted in a buffer queue sorted on this calculated distance in order to be searched. However, upon reaching a leaf, not *minDist* but the actual distance $d$ is calculated. If the latter value is smaller than the one calculated so far, then it is introduced as the new minimum distance. This minimum distance is used in pruning subtrees, which correspond to a higher optimal distance (more details and explanations can be found in [23]).

To generalize this query to the $k$-nearest neighbors query, a descending heap of size $k$ is used in order to store the $k$ most similar images to the queried image found so far, along with a buffer that stores the nodes that still have to be searched. While descending the tree, when visiting an internal node its children are stored as previously into the second buffer based on the respective $minDist$. There is a slight difference in this case though at the leaf level. Upon reaching a leaf, as previously not the optimal but the actual distance $d$ is calculated and, in case it is smaller than the $k$-th distance found so far, it is inserted in the heap containing the $k$ smaller calculated distances and it is stored in the appropriate position. In addition, the distance that is used to prune subtrees which represent a higher optimal distance is the greatest value stored in the heap. Evidently, the buffer is initialized containing the infinity in all $k$ positions.

# 5 Experimental Results

## 5.1 Experimental Setup

In order to evaluate the effectiveness and efficiency when using the proposed image signatures and the S-tree we realized a large number of experiments using real image sets. A set of 50,000 images (merged from two commercially available collections: *PrintArtist Platinum* (nearly 20,000 images) by Sierra Home and some *Master Photos 50,000 – Premium Photo Collection* (about 30,000 images) by COREL, was divided in 5 sets of 10,000, 20,000, ... 50,000 images – hereafter referred to as the S10k, S20k, ..., S50k respectively – and those were used as the image database. The query images were obtained from yet another collection of images (*Gallery Magic 65,000* by COREL) to diminish the probability of biasing the results obtained. Each query image is a constituent of a subset determined (also *a priori*) of similar images[2] in order to allow us to assess retrieval accuracy objectively. The images in each of these subsets tend to resemble each other based on the color distribution and also the image semantics (all subsets can be seen at: `http://www.cs.ualberta.ca/∼mn/CBIRdataset/`). The color constituents were normalized to a 64 color representation using the RGB color model. We only report results when using VBA with $t = 10$; experiments using signatures where all bins had the same capacity showed that allocation was just as effective as the traditional GCH. Even though one could argue that such an approach would still be much more compact than GCH we decided to focus on the VBA scheme which is equally compact and yields better retrieval. Results using the GCH and Color Coherence Vectors (CCV) [22] were also generated to serve as a benchmark to our proposal's effectiveness.

The results presented next are divided in two main parts. In the first, retrieval effectiveness is measured through the use of precision vs recall curves [37], which are traditionally used in the Information Retrieval literature. At that point the indexing structure is irrelevant. Next we present results related to efficiency of retrieval. As such we report figures for storage overhead and query processing time. Given that our similarity metrics are quite simple we consider query processing an I/O bound process and therefore we use the number of I/Os required as a measure of query processing time. In particular, we measure the number of leaf nodes accessed in order to avoid any side-effects from caching of internal nodes. In the latter part we investigate the influence of several parameters as well.

## 5.2 Retrieval Effectiveness

Figure 3 shows the precision and recall yielded by the use of GCH, CCV and VBA using 30,000 images. Note that at this point the performance of the underlying index is irrelevant.

To show the competitiveness of VBA for different sizes of the dataset, Figure 4 shows the precision values obtained for all five sets at 20% recall (e.g., assuming that most users will look only at the first few matches returned). As one can see, for all approaches, precision decreases steadily with the increase in the dataset size, which is a reasonable result. Due to limited space we do not show similar figures for other values of recall, but we have observed that the quantitative behavior does not change as much. Overall we have found that with respect to retrieval accuracy, VBA and CCV have similar performance, with no clear winner. It is clear nonetheless that both outperform GCHs.

## 5.3 Storage Overhead

We investigated the space requirements of our approach with respect to two variables: disk page size (4, 8 and 16 Kbytes) and the size of the image set. We compared the size of the SR-tree indexing GCHs and CCVs (denoted

[2] At this point it is noteworthy pointing out that, to the best of our knowledge, there is no standard benchmark collection for effectiveness evaluations of different CBIR approaches, thus the *ad-hoc* nature of our evaluation method.

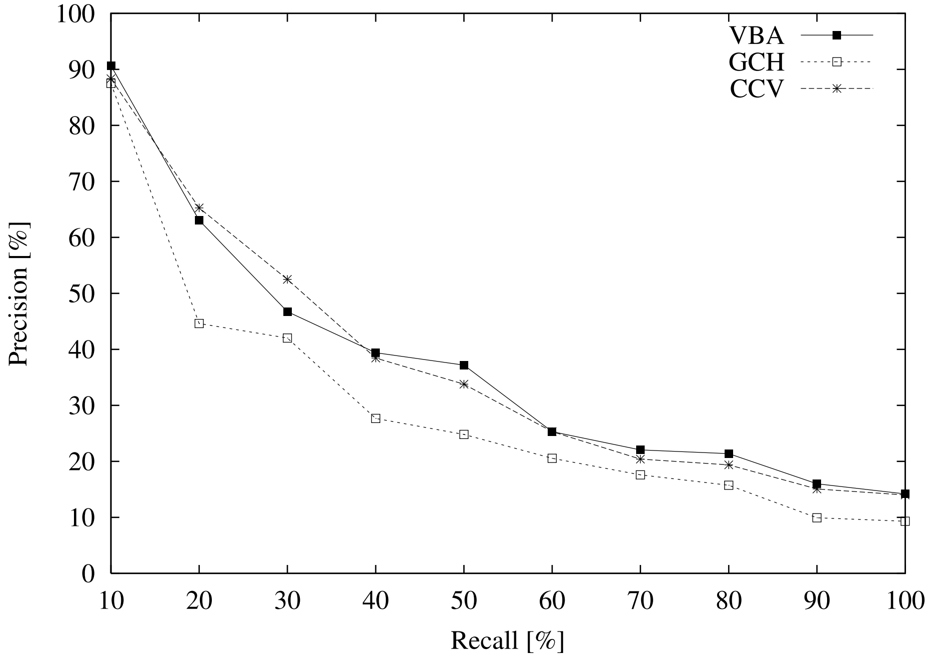


Figure 3: Precision and Recall for dataset size of 30,000 images.

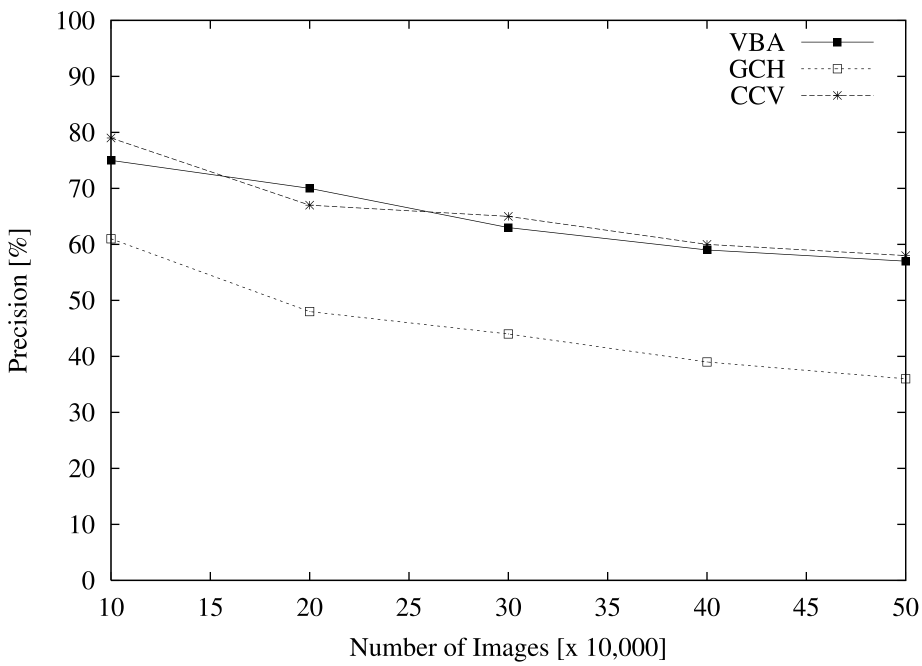


Figure 4: Precision at 20% Recall for all dataset sizes.

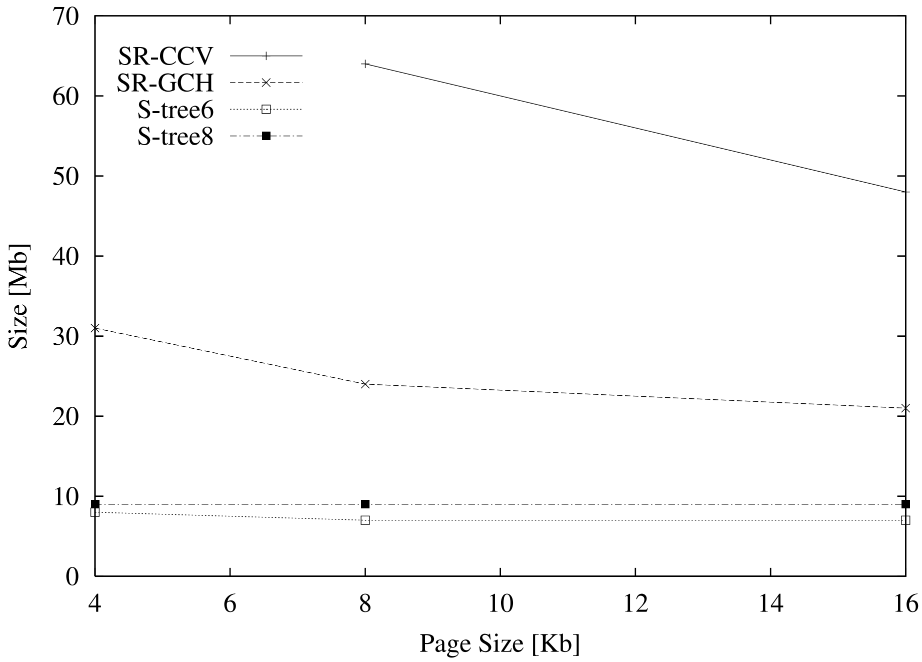


Figure 5: Index (or file) size for different page sizes.

by SR-GCH and SR-CCV, respectively) to the size of the S-tree indexing VBA signatures using logical pages of approximately 600 and 800 bytes (denoted S-tree6 and S-tree8 respectively).

At this point it is important to recall that an image's CCV is twice as large (in terms of dimension) than its GCH. This is so, because, for each color, it records the ratio of coherent and incoherent pixels. As such one should expect the SR-tree indexing CCVs to be at least twice as large as an SR-tree indexing GCHs for the same dataset. This also has direct implication on the SR-tree's fan-out factor, yielding a tall and narrow tree (instead of a shallow and wide one). This is clear from Figure 5. This figure was obtained using 30,000 images and it indicates that, while the SR-tree takes advantage of larger page sizes (which increase its nodes' fan-out), it is still over two times larger than the S-tree8 for a page size of 16 Kbytes. More importantly however, it shows clearly that the SR-tree is not able to help CCV's very high-dimensionality, and therefore it results in a very large indexing file.

We did not use a page size of 4K for the SR-tree/CCV combination because it did not seem to be a practical choice.

In order to evaluate the effect of the size of the image set on the access structures, we kept the page size fixed at 8 Kbytes, and indexed all image sets (S10k through S50k). Figure 6 shows that the SR-tree grows much faster than the S-tree. While the S-trees do grow with the enlargement of the image sets, the curve is not nearly as steep. At its most compact configuration (10 Kbytes), the SR-tree/GCH was almost 3 times as big as the S-tree8. Again, the SR-tree/CCV combination grows extremely fast with the increase of the dataset size, and further values were not computed.

We thus conclude that storage-wise the S-trees are more efficient than the SR-tree indexing GCH. Due to the very high-dimensionality of the CCVs their indexing, even using an efficient access structure, in this case the SR-tree, did not seem to be a feasible alternative.

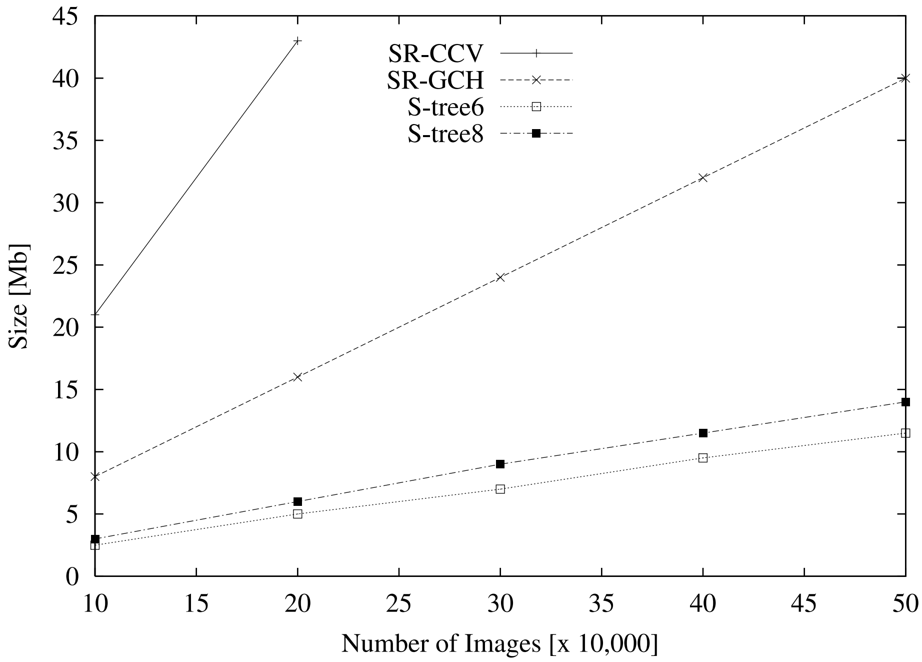


Figure 6: Index (or file) size for all dataset sizes.

### 5.4 Retrieval Efficiency

We have observed thus far that the VBA approach is comparable to CCV's in terms of retrieval effectiveness. Both VBA and CCV offer better performance than the GCH. In terms of size, S-trees indexing VBA signature are by far more compact than SR-trees indexing GCHs, which in turn are less than half the size of SR-trees indexing CCVs. In this section we compare the performance of the SR-tree/GCH, SR-tree/CCV and S-tree/VBA combined approaches. We investigated the query processing time required by all approaches using the number of leaf nodes accessed, and three variables: disk page size, the size of the image set and number of similar matches returned.

Figure 7 shows how each investigated combination behaves when the page size changes. The number of indexed images was kept constant at 30,000 and the number of matches returned (nearest neighbors) was set to 20. Clearly the SR-tree/CCV is the slowest combination of all tested, while the SR-tree6/VBA was the fastest. All combinations seemed to be equally affected (positively) by the increase in the page size. While the S-tree8 seemed to be slightly faster than the SR-tree8/GCH, the S-tree6 was, in average, 20% faster than the latter.

We can see the effect of the number of indexed images in Figure 8. For these experiments, the page size was constant, 8 Kbytes, and again, the number of similar images returned was set to 20. The results seem to indicate that the number of I/Os by the SR-tree/GCH curve grows slightly faster with the data set size than the combination using S-trees. The gain from using the S-tree8 (which is slower than the S-tree6) ranged in the neighborhood of 15%. Again, the SR-tree/CCV is not only the worse combination at the outset, but it becomes slower faster (in fact, it was not used for large datasets).

All combinations but the SR-tree/CCV seemed to be equally affected when the number of best matches returned varied, as one can verify in Figure 9. Clearly, the SR-tree/CCV combination is not practical when compared to the other ones. The S-tree6/VBA combination offered consistently the best performance, while the S-tree8/VBA performed slightly better than the SR-tree/GCH.

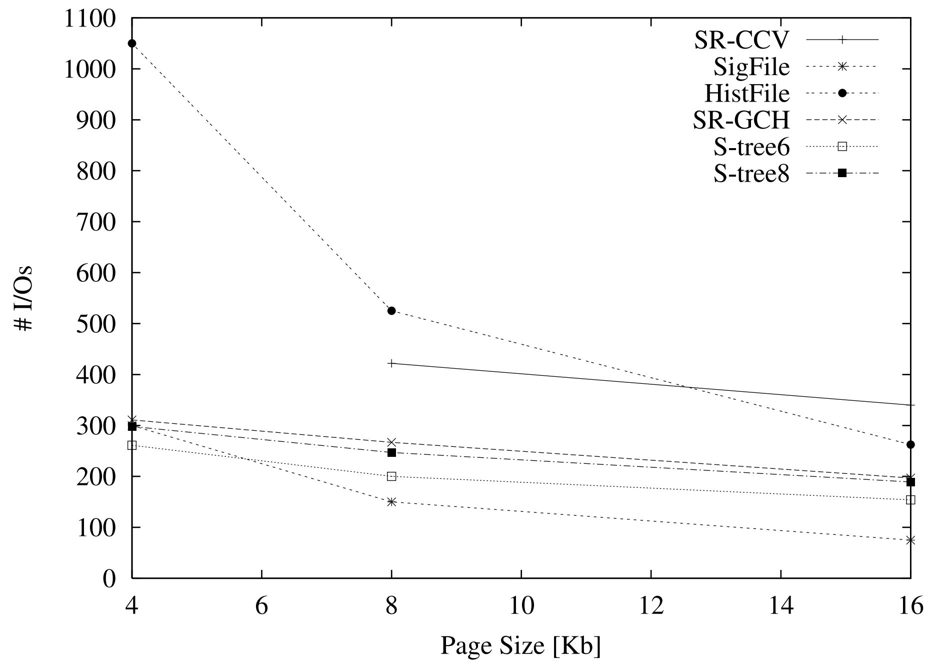


Figure 7: Query time (# I/Os) for different page sizes.

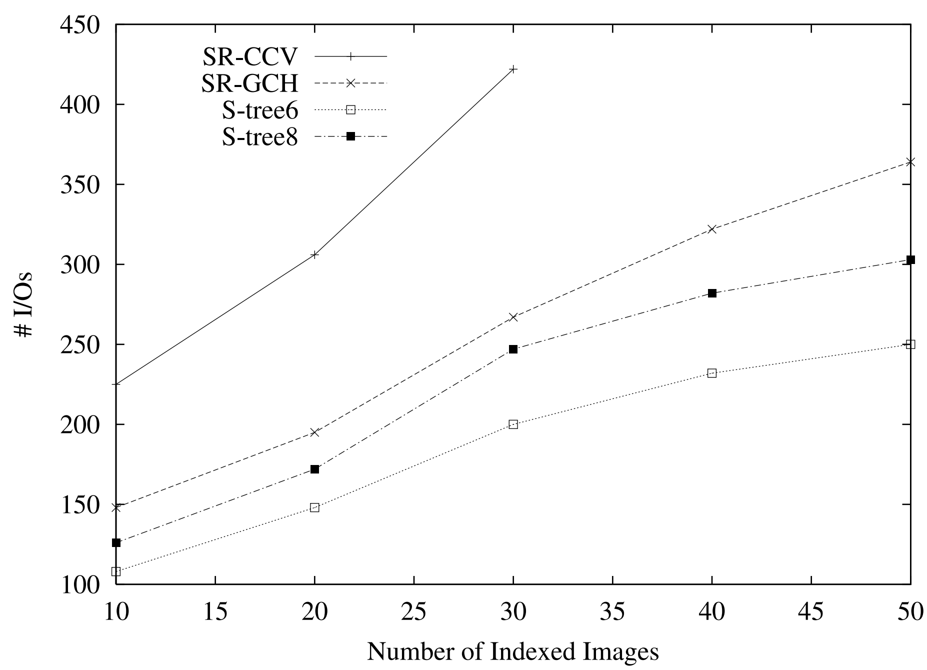


Figure 8: Query time (# I/Os) for all image set sizes.

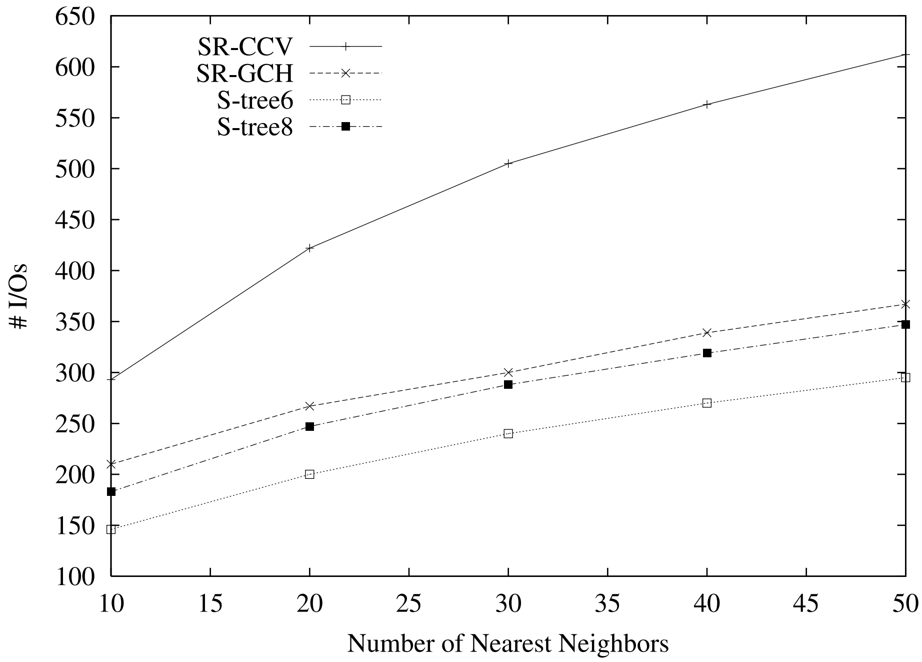


Figure 9: Query time (# I/Os) for different answer sizes.

# 6 Conclusions

We offer two main contributions in this paper: (i) a new way to represent an image's color distribution via binary signatures (called VBA) along with a metric to compute similarities between images; and (ii) an efficient, space and speed-wise, method (the S-tree) to index such image signatures, and the corresponding algorithm for nearest-neighbor (similarity) queries.

While the use of VBA signature yields retrieval accuracy comparable to CCVs (though much better than GCHs), our experimental results have shown that the combination CCV/SR-tree is not a practical one due to the large space overhead imposed by the CCVs. Clearly, not using any indexing at all is not an option either. On the other hand the VBA/S-tree combination outperforms consistently the GCH/SR-tree combination for all possibilities of page size, dataset size and answer size we investigated. Therefore we claim the VBA/S-tree combination to be a robust, effective and efficient approach to the CBIR problem, in particular for the case when one is interested in the images' color distribution.

Two main issues remain open for future research. One is the point (in number of images) where a simple linear scan of the signature file would be more efficient than using an indexing structure. Even though an index does speed up query processing for large datasets, it also imposes some overhead which may not pay off for small datasets. The second issue is related to the bit allocation scheme, which is itself a variable. Although the scheme detailed in this paper has given good results, we do not claim it is optimal. In fact, we are further investigating other possibilities.

# Acknowledgments

We gratefully acknowledge the use of the SR-tree provided by N. Katayama and S. Satoh. We would also like to thank R.O. Stehling for organizing the dataset which was used for evaluating the proposed retrieval methods. M.A. Nascimento was partially supported by a Research Grant from NSERC, Canada. Y. Manolopoulos' work

was performed while at the Computer Sc. Dept. of the Univ. of Cyprus.

## References


[1] A. R. Appas et al. Image indexing using composite regional color channels features. In *Proc. SPIE - Storage and Retrieval for Image and Video Databases VII*, volume 3656, pages 492–500, 1999.

[2] N. Beckmann et al. The R$^*$-tree: an efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD'90 Conf.*, pages 322–331, 1990.

[3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: an index structure for high-dimensional data. In *Proc. 22nd VLDB Conf.*, pages 28–39, 1996.

[4] V. Chitkara, M. A. Nascimento, and C. Mastaller. Content-based image retrieval using binary signatures. Technical Report TR-00-18, Dept of Computing Science, University of Alberta, 2000.

[5] S. Christodoulakis and C. Faloutsos. Signature files: an access method for documents and its analytical performance evaluation. *ACM Transactions on Office Information Systems*, 2(4):2671–288, 1984.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. 23rd VLDB Conf.*, pages 426–435, 1997.

[7] N. G. Colossi and M. A. Nascimento. Benchmarking access structures for high-dimensional multimedia data. In *Proc. IEEE Multimedia Conf. and Expo*, pages 1215–1218, 2000.

[8] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[9] A. del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann, 1999.

[10] U. Deppisch. S-tree: a dynamic balanced signature index for office retrieval. In *Proc. 9th ACM SIGIR Conf.*, pages 77–87, 1986.

[11] C. Faloutsos. Signature files. In *Information Retrieval: Data Structures and Algorithms*. W. B. Frakes and R. Baeza-Yates (eds), Prentice Hall, 1992.

[12] M. Flickner et al. Query by image and video content: The QBIC system. *IEEE Computer*, pages 23–32, 1995.

[13] V. Gaede and O. Guenther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[14] S. Helmer and G. Moerkotte. Evaluation of main memory join algorithms for joins with set comparison join predicates. In *Proc. 23rd VLDB Conf.*, pages 386–395, 1997.

[15] W. Hsu, T. S. Chua, and H. K. Pung. An integrated color-spatial approach to content-based image retrieval. In *Proc. 3rd ACM Multimedia Conf.*, pages 305–313, 1995.

[16] IBM. IBM's Query by Image Content. `http://wwwqbic.almaden.ibm.com/`.

[17] Y. Ishikawa, H. Kitagawa, and N. Ohbo. Evaluation of signature files as set access facilities in oodbs. In *Proc. ACM SIGMOD'93 Conf.*, pages 247–256, 1993.

[18] N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *Proc. ACM SIGMOD'97 Conf.*, pages 369–380, 1997.

[19] S. Lin. An extendible hashing structure for image similarity searches. Technical Report TR-00-06, Dept of Computing Science, University of Alberta, 2000. (MSc Thesis).

[20] W. Niblack et al. The QBIC project: Querying images by content using color, texture and shape. In *Proc. SPIE - Storage and Retrieval for Image and Video Databases*, pages 173–187, 1993.

[21] D.-S. Park et al. Image indexing using weighted color histogram. In *Proc. 10th Conf. on Image Analysis and Processing*, 1999.

[22] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *Proc. 4th ACM Multimedia Conf.*, pages 65–73, 1996.

[23] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD'95 Conf.*, pages 71–79, 1995.

[24] Y. Rui, T. S. Huang, and S.-F. Chang. Image retrieval: Past, present, and future. *J. of Visual Communication and Image Representation*, 10(1):39–62, 1999.

[25] E. D. Sciascio, G. Mingolla, and M. Mongiello. Content-based image retrieval over the web using query by sketch and relevance feedback. In *Proc. 4th Conf. on Visual Information Systems*, pages 123–130, 1999.

[26] J. R. Smith and S.-F. Chang. Tools and techniques for color image retrieval. In *Proc. SPIE - Storage and Retrieval for Image and Video Database IV*, pages 40–50, 1995.

[27] J. R. Smith and S.-F. Chang. Visually searching the web for content. *IEEE Multimedia*, 4(3):12–20, 1997. (`http://www.ctr.columbia.edu/webseek`).

[28] R. O. Stehling, M. A. Nascimento, and A. X. Falcao. On 'shapes' of colors for content-based image retrieval. In *Proc. Workshop on Multimedia Information Retrieval*, pages 171–174, 2000.

[29] M. Stricker and M. Orengo. Similarity of color images. In *Proc. SPIE - Storage and Retrieval for Image and Video Databases III*, pages 40–50, 1995.

[30] M. J. Swain and D. H. Ballard. Color indexing. *Computer Vision*, pages 11–32, 1991.

[31] E. Tousidou, P. Bozanis, and Y. Manolopoulos. Efficient handling of signature files used for objects with set-valued attributes. 2000. Submitted for publication.

[32] E. Tousidou, A. Nanopoulos, and Y. Manolopoulos. Improved methods for signature-tree construction. *The Computer Journal*, 43(4):301–314, 2000.

[33] C. Traina et al. Slim-trees: High performance metric trees minimizing overlap between nodes. In *Proc. 7th EDBT Conf.*, pages 51–65, 1999.

[34] Virage. VIR image engine. `http://www.virage.com/products/vir-irw.html`.

[35] J. Z. Wang. SIMPLIcity: a region-based image retrieval system for picture libraries and biomedical image databases. In *Proc. 8th ACM Multimedia Conf.*, pages 483–484, 2000. (`http://wang.ist.psu.edu/cgi-bin/zwang/regionsearch_show.cgi`).

[36] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. 12th ICDE Conf.*, pages 516–523, 1996.

[37] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.