Selecting Good Keys for Triangle-Inequality-Based Pruning Algorithms

Andrew Berman and Linda G. Shapiro *

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

{aberman,shapiro}@cs.washington.edu

Abstract

A new class of algorithms based on the triangle inequality has recently been proposed for use in content-based image retrieval. These algorithms rely on comparing a set of key images to the database images, and storing the computed distances. Query images are later compared to the keys, and the triangle inequality is used to speedily compute lower bounds on the distance from the query to each of the database images. This paper addresses the question of increasing performance of this algorithm by the selection of appropriate key images. Several algorithms for key selection are proposed and tested.

1 Introduction

There is a growing need for the ability to query image databases based on image content. Such content-based retrieval usually revolves around the use of a distance measure–a scoring function that rates the similarity of two images. Since distance measure computation can be expensive, there is a need to eliminate candidate database images from a search without direct comparison to the query image. One method is to eliminate candidate images by the use of the triangle inequality[3]. In this scheme, a set of *key images* is stored in the database, along with the distance from each database image to each key image. When the user chooses a query image, the distance from the query image to each key image is computed. The triangle inequality can then be used to compute lower bounds on the distance from the query image to each of the database images, possibly eliminating database images as candidates for an approximate match to the query. The number of key images is assumed to be much smaller than the number of database images–thus, the total number of distance computations is greatly reduced. This scheme can be expanded to the use of multiple distance measures in combination.

1.1 Selecting Good Keys

One aspect of triangle-inequality-based search algorithms previously unaddressed in the literature is how to select keys which maximally eliminate images without direct comparison. This paper proposes and tests several algorithms for selecting good keys.

1.2 Fundamental Query Strategies: Threshold and Best-Match

Given an image database S, a query image q, and a distance measure d, there are two main types of searches. One can request all images $s \in S$ such that the distance $d(q, s)$ is less than some threshold value T. We label this task Threshold. The second search task is to find the image or images in S which minimize $d(q, s)$. We label this task Best-Match. In this paper, we evaluate key selection algorithms for triangle-inequality-based searches for both Threshold queries and Best-Match queries.

2 Overview of the Triangle Inequality Algorithm

There are several schemes in the literature that take advantage of the triangle inequality to reduce the number of direct comparisons in a Threshold search[3, 4, 2, 6, 1]. The intuition behind all the schemes is that the distance between two objects cannot be less than the difference in their distances to

*This research was supported by the National Science Foundation under Grant No. IRI-9711771 and by the Washington Technology Center.

any other object. More formally, let i represent a database object, q represent a query object, k represent some *key* object, and d represent some distance measure that is a metric. The following inequality is always true:

$$d(i,q) \geq |d(i,k) - d(q,k)| \tag{1}$$

Thus, by comparing the database and query objects to a third *key* object, a lower bound on the distance between the two objects can be obtained. We define $l(d,k,i,q)$ to be equal to the lower bound on $d(i,q)$ found by calculating $|d(i,k) - d(q,k)|$. We further shorten $l(d,k,i,q)$ to $l(d,k)$ when there is no confusion as to the values of i and q.

Burke and Keller[6] first proposed the idea of using the triangle inequality to reduce comparisons. Uhlmann[10] described the ***vantage point tree***, or vp-tree, in which relative distances to key objects are repeatedly used to partition the database. Each node in the tree represents a vantage-point. The left child containing those objects which are within a certain distance to the vantage point, while the right child contains the remaining objects. By repeatedly comparing the query to the vantage-point objects, one can eliminate subtrees from the search. Enhancements to the vp-tree include partitioning of the distances into more than two sets, and multiple vantage points at each node [5].

In a vantage point tree, each node traversed requires a distance comparison between the query and the key. Berman[3] and Baeza-Yates, et. al.[1] invented a tree structure in which all the nodes at a given level are associated with a single key. Thus, the total number of distance comparisons required while traversing the tree is only the height of the tree. Individual node traversals are thus very cheap. The nodes are assumed to have many more than 2 children, since it is worth pruning as many leaves as possible at the cost of having extra node traversals.

Barros, et. al.[2], successfully used a single set of keys and the triangle inequality in a real image database. They did not use any tree structure, but stored all the distances in a table. In Berman and Shapiro[4] we introduced an indexing and retrieval algorithm for multiple keys and multiple distance meaures, and stored the distances in a table, rather than a tree structure. This algorithm is summarized below.

2.1 Threshold Searches with Multiple Keys

Equation (1) can be extended naturally by substituting a set of keys $K = (k_1, \ldots, k_M)$ for k as follows:

$$d(i,q) \geq max_{1 \leq s \leq M} |d(i,k_s) - d(q,k_s)| \tag{2}$$

We define $l'(d,K,i,q)$ to be equal to the lower bound on $d(i,q)$ found by using equation 2. As before, we shorten $l'(d,K,i,q)$ to $l'(d,K)$ where possible.

Consider a large set of database objects, $I = \{i_1, \ldots, i_N\}$ and a much smaller set of key objects, $K = \{k_1, \ldots, k_M\}$. Pre-calculate $d(i_s, k_t)$ for all $\{1 \leq s \leq M\} \times \{1 \leq t \leq N\}$. Now consider a request to find all database objects i_s such that $d(i_s, q) \leq T$ for some query image q and threshold value T. We can calculate lower bounds on $\{d(i_1,q), \ldots, d(i_N,q)\}$ by calculating $\{d(q,k_1), \ldots, d(q,k_M)\}$ and repeatedly using equation (2). If we prove that T is less than $d(i_s,q)$, then we eliminate i_s from our list of possible matches to q. After the elimination phase, we search linearly through the uneliminated objects, comparing each to q in the standard fashion. This algorithm involves $M + U$ distance measure calculations, and $O(MN)$ simple (constant cost) operations, where U is the number of uneliminated objects. The hope is that $M + U$ is sufficiently smaller than N to result in an overall time savings.

2.2 Threshold Searches with Multiple Distance Measures

In Berman and Shapiro[4], we extended the above scheme to work with combinations of distance measures. The intuition is that lower bounds on the distance between two objects for distance measures d_1 and d_2 can be used to calculate a lower bound between the objects for distance measure d when d can be calculated directly from d_1 and d_2.

Let $D = d_1, \ldots, d_P$ be a set of distance measures. These distance measures will be known as the *base* distance measures. Let $K = K_1, \ldots, K_P$ be a set of sets of keys. Let $L(D,K,i,q)$ be the set of lower bounds $l'(d_s, K_s, i, q)$ calculated from equation 2 for each tuple $(d_s \in D, K_s \in K), 1 \leq s \leq P$.

Now consider a new distance measure d' that is of the form $d'(i,q) = f(d_1(i,q), \ldots, d_s(i,q))$, where f is monotonically non-decreasing in its parameters. For example, f might describe a weighted sum of the base measures, or even combinations of minima and maxima of sets of the base measures. Since $l'(d_s, K_s, i, q) \leq d_s(i, q)$ for all s, substituting $l'(d_s, K_s, i, q)$ for each instance of $d_s(i, q)$ gives us $d'(i, q) \geq f(l'(d_1, K_1, i, q), \ldots, l'(d_s, K_s, i, q))$. Thus we can calculate a lower bound on $d'(i, q)$ given lower bounds on the base distance measures. As with single distance measures, we can thus eliminate database objects as candidates for approximate matching to a query object without direct comparisons.

2.3 Implementing Best-Match Using the Triangle Inequality

The calculated lower bounds on the distance from the database objects to the queries have an interesting property that we are currently attempting to exploit. Suppose we are given two database images i_1 and i_2 and query q such that $d(i_1, q) < d(i_2, q)$. Experimental evidence suggests that quite often $l(d, k, i_1, q) < l(d, k, i_2, q)$. That is, the ordering of the lower bounds reflects the ordering of the closeness of the images to the query. The correlation between the two inequalities increases with the number of keys. In tests of up to 1800 images with multiple distance measures, we have found that the best match to a query can often be determined by ordering the images on the basis of their calculated lower bounds and directly examining the first dozen or so images.

3 What is a Good Key?

We begin this section with a discussion of what makes a good single key. Later, we discuss the choice of keys in combination. We make the simplifying assumption that all distances are within the range of 0 to 1, inclusive. We also make the assumption that the database is static and known in advance.

3.1 Good Keys for Threshold Style Queries

Consider database image i, query image q, key image k, distance function d. We say that key k *separates* q from i for value v if $|d(i, k) - d(q, k)| > v$. Suppose that $d(i, q) > T$ for some threshold T. The triangle inequality implies that the value $|d(i, k) - d(q, k)|$ can range from 0 to $d(i, q)$. Key k will eliminate image i as a candidate match to q only if it separates i from q for value T. The purpose of the algorithm is to eliminate as many non-matching candidate images as possible through key comparison. Thus, a good key will eliminate more candidate images than a poor key. The concept of separation described above motivates the following discussion.

Given a set of database images S, distance measure d, and key k, we can compute a density function f on $d(s, k), s \in S$. Since we do not know the queries in advance, we make the simplifying assumption that the queries are taken from the database images and ignore exact matches in our searches. Given threshold T, we can calculate the fraction of images that k will separate from a random query by looking at this density function. For example, if all of the area of the density function lies in a narrow range $(x, x + e), e < T$, as shown in Figure 1a, then k will never separate any query from any image in the database. If the density function has a uniform distribution, as shown in Figure 1b, then for $0 < T < 1/2$, $P(k$ separates i from $q) = (1 - T)^2$. If the density function is multipolar, with N equally sized narrow spikes separated by distance greater than T, as shown in Figure 1c, then $P(k$ separates i from $q) = (N - 1)/N$. If the density function has a Gaussian shape, as shown in Figure 1d, then, roughly speaking, greater standard deviations will indicate greater average separation of images by the key.

The issue gets more complicated when choosing several keys. Using keys k_1 and k_2 will be no better than just using k_1 if they both separate the same images from queries. The question of whether or not two keys separate the same images is computationally expensive to answer in the general case, but can be approximately answered by sampling. One can also use the fact that very similar keys will separate the same images and thus try to avoid keys that are too close together. For example, in a clusterable database, keys should come from different clusters. Indeed, the key selection algorithms with the best results make use of clustering and ensuring that different keys separate different images.

3.2 Good Keys for Best-Match Queries

Given images i_1 and i_2, query q and key k, assuming that $d(i_1, q) < d(i_2, q)$, key k *orders* i_1 and i_2 correctly if $l(d, k, i_1, q) < l(d, k, i_2, q)$. We can extend this definition naturally to sets of keys and multiple distance measures. Although our analyses were for Threshold queries, the results were very good for Best-Match queries as well. Further analysis of keys optimized for Best-Match queries is an open area of research.

4 Algorithms for Key Selection

We examined five different algorithms for key selection: random keys, choosing keys by examining

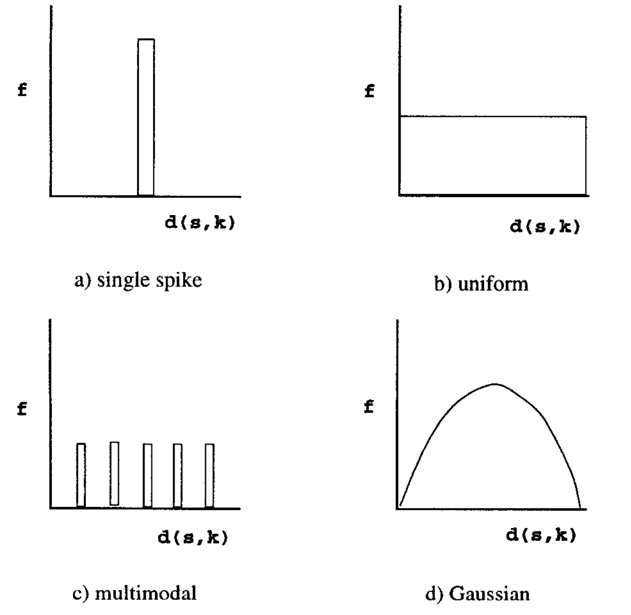

Figure 1: The shape of the density functions determines the performance of the keys.

the variance of the density function, ranking by testing thresholding efficiency, a greedy thresholding algorithm, and a clustering algorithm. The algorithms assume a database S and a set of candidate keys.

Random Our prototype image database system currently uses a set of (up to 20) keys chosen randomly and uniformly from the database itself. The triangle inequality algorithms give excellent performance compared to linear search even with random keys, so this is a natural benchmark against which to test the other algorithms.

Variance Taking a subset S' of our database S, we calculated the density function of $d(k,s), s \in S'$ for each candidate key k. We selected those candidate keys which had the density functions with the greatest variance.

Separation We examined our database by hand to find pairs of images that we judged to be approximate matches. The average distance between these pairs was calculated. This value T represented a potential "threshold value" that one might use in a query to find approximate matches. We then selected those candidate keys k which maximized $P(|d(s_1,k) - d(s_2,k)| > T)$ over all pairs $s_1, s_2 \in S'$, where S' was a subset of our database S.

Greedy Variance and Separation may choose several keys which separate the same pairs of images. We thus modified Separation to keep track of which pairs of images were separated by each key. The first key selected was the same as that selected by Separation. The performance of the remaining keys were then recalculated to discount pairs of images already separated by the first key. This process was continued for subsequent keys until a preset number of keys was selected.

Cluster We used a simple clustering algorithm on the database. We selected the two database images k_1 and k_2 that were furthest apart, and used them as initial seeds for clustering. These two images were placed into our set of keys, and the remaining images were assigned to clusters based on their distances to the key images. We then found the image that was furthest from the current set of keys, added it to the set, and re-clustered the database on the updated set of keys. We continued this process until the correct number of keys were selected.

5 Experiments

For our experiments, we collected two sets of images, one with 600 members, and one with 800 members. From each set, 100 images were chosen arbitrarily to be candidate key images. The remaining 500 and 700 images became the test database. The five algorithms were run on the candidate images to choose sets of 1 to 9 keys.

We queried the database against itself testing the system's performance using the keys chosen by the key selection algorithms. To eliminate exact matches, we temporarily removed each query image from the database. To test the performance of the keys on a Best-Match search, we determined the best match to the query and calculated its position in the ordering of the lower bounds. To test the performance of the keys on a Threshold search, we counted the number of images separated from the query by a given threshold value. This threshold value was determind off-line by calculating the average distance between pairs of images known to be similar. For the Random key selection algorithm, we ran the tests 10 times and averaged the results.

5.1 The Distance Measures

Our prototype database system allows great flexibility in forming composite distance measures from a set of "base" distance measures. For our tests, we used two base distance measures: a color measure and a texture measure. We then modified both of them to add spatial locality, creating four more measures. We also tested two composite distance measures, containing both texture and color components . A short summary of the various distance measure follows:

Color Histogram This is a simple L_1 distance measure based on breaking up the color space into a 4×4×4 RGB cube.

Local Binary Partition Texture This is a standard and easy to implement texture measure with very good performance[11]. For each pixel p, the 8 neighbors are examined to see if their intensity is greater than p. The result becomes an 8 digit binary number, and a histogram of the numbers is created for each image. Two images are compared by taking the L_1 distance between their histograms.

Horizontal Color and Horizontal Texture For these two distance measures, each image was split into three equal-sized horizontal pieces. Two images were compared by averaging the Color or Texture distance between the corresponding pieces.

Vertical Color and Vertical Texture These distance measures are similar to the horizontal distance measure above, except that the images were split vertically.

Minimum(Color, Texture) Our prototype system allows taking the minimum of several distance measures. This is useful in cases where the user wishes to find a similar image without detailing the nature of the correspondence. For this measure, the color distance and texture distance are both calculated, and the minimum returned.

Sum(Color, Texture) Several image retrieval systems such as QBIC and Virage offer the weighting of color, texture, and other qualities. This measure is equivalent to an equal weighting of color and texture.

6 Results

We discuss the performance of the various key selection algorithms, first for Best-Match and then for Threshold. As the rankings of the algorithms did not change much as a function of number of keys, we only show the results for 9 keys, the maximum number tested. As the performance of the algorithms on the

A	B	C	D	E	F	G
Color Histogram	GR	0.9%	CL	1.1%	1.6%	1.8
LBP Texture	CL	1.2%	GR	1.6%	2.3%	1.9
Horizontal Color	GR	3.6%	CL	3.7%	4.6%	1.3
Vertical Color	GR	2.5%	RA	3.5%	3.5%	1.4
Horizontal Texture	CL	2.4%	GR	3.7%	4.8%	2.0
Vertical Texture	CL	2.1%	GR	2.2%	3.9%	1.9
Min(Color, Texture	GR	1.0%	CL	1.7%	2.2%	2.2
Color + Texture	GR	1.0%	CL	1.1%	1.7%	1.7

Table 1: Best Algorithms for Best-Match with 9 keys on a database of 500 images

Column Headings:
A: Distance measure
B: Best key selection algorithm: GR=Greedy, CL=Cluster, RA=Random
C: Average rank of best match using best algorithm
D: Second best key selection algorithm
E: Average rank of best match using second best algorithm
F: Average rank of best match using randomized key selection
G: Ratio of performance of randomized algorithm to best algorithm (F/C)

two databases was very similar, we only show tables for the larger database.

6.1 Performance of Key Selection Algorithms for Best-Match

As is shown in Table 1, Cluster provided the best keys for the texture measures, while Greedy provided the best keys for the color measures and the combination color/texture measures. The second best algorithm was also always Greedy or Cluster except for the *vertical texture* measure in the larger database, which had Random as the second best.

Columns C, E, and F of the table show the average rank of the best match using the appropriate key selection algorithm. For example, a 2% would mean that the true best match was ranked in the top 2% of the returned images. Column G represents the ratio of the number of images which would be examined using the Random keys to the number of images which would be examined using the best discovered keys. Thus, in the first row, the best keys returned the closest match in the top 0.9% of the images. For the 700 image database, this translates to the top 6 or 7 images. The random keys returned the closest match in the top 11 images. If this database was a representative sample of a 700,000 image database, then the number of images needed to be directly compared would be approximate 630 and 1120 respectively. On average, there was a 42% reduction in the number of images examined using the best discovered keys compared to using the random keys.

The overall performance of the algorithms was excellent. In the worst case for the database of 700 images, *horizontal color*, the closest match was ranked in the top 3.6%–that meant that only approximately 24 images had to be compared directly after pruning to find the best match. The database of 500 images had slightly worse performance with the average ranking of the best image ranging from 1.2% to 5.1%, again with the worst performance found in *horizontal color*.

6.2 Performance of Key Selection Algorithms for Threshold

As table 2 shows, the Greedy key selection method was the clear winner for Threshold, yielding the best performance for every distance measure. There was no clear second place algorithm–Random, Variance, and Cluster all appeared in second place for several measures. The Greedy keys reduced the number of images that had to be directly compared by the Random keys by 16% to 44%.

The second thing to note in Table 2 is the wide range of performance between distance measures. The triangle inequality algorithm thresholded 98.5% of the images for the *Color Histogram* distance measure, yet it only thresholded 54.6% of the images for the *Vertical Texture* distance measure. It is difficult to compare across distance measures since the distribution of distance values across pairs of images vary greatly from measure to measure. Especially interesting was the fact that using 5 keys for *Vertical Texture* resulted in a 53% thresholding. Thus, the additional four keys only eliminated an additional two percent of the database. In [1], Baeza-Yates, et. al.., demonstrated how, given a random model for database objects and keys, a logarithmic number of keys should threshold almost all of the database. For our experiments to have supported

A	B	C	D	E	F	G
Color Histogram	GR	98.5%	RA	97.4%	97.4%	1.7
LBP Texture	GR	83.2%	CL	81.2%	77.8%	1.3
Horizontal Color	GR	94.4%	RA	90.8%	90.8%	1.6
Vertical Color	GR	93.2%	VA	90.0%	89.8%	1.5
Horizontal Texture	GR	55.3%	CL	49.9%	47.5%	1.2
Vertical Texture	GR	54.6%	VA	52.7%	47.2%	1.2
Min(Color, Texture)	GR	97.8%	CL	96.4%	96.1%	1.8
Color + Texture	GR	94.5%	CL	91.5%	91.2%	1.6

Table 2: Best Algorithms for Threshold with 9 keys on Database of 700 images

Column Headings:
A: Distance measure
B: Best key selection algorithm: GR=Greedy, CL=Cluster, RA=Random, VA=Variance
C: Average percent of database eliminated using best algorithm
D: Second best key selection algorithm
E: Average percent of database eliminated using second best algorithm
F: Average percent of database eliminated using randomized key selection
G: Ratio of performance of randomized algorithm to best algorithm $(100\% - F)/(100\% - C)$

this, the addition of four more keys would have had to increase the thresholding from 53% to about 70%. That this didn't occur demonstrates that traditional models of randomness do not really apply to images or groups of images.

7 Future Work

The performance of keys in image retrieval is intimately tied to the statistical behavior of the distance measures over the image set. At present, we have a limited understanding of this behavior; this limits the sophistication of our key selection algorithms. Thus, more research into the behavior of the distance measures is called for. A more complete set of distance measures will also be used in our future tests. Distance measures have been proposed for color, texture, shape[8], object presence[9], and object spatial relationships[7]. We would like to include representatives of each type of measure in our tests.

We assumed a static database known in advance of the key selection. Many databases do not have these qualities. Finding good keys for non-static databases using triangle inequality algorithms is an open research problem.

In our work, we selected keys from the database itself. The space of possible keys is huge– it is the space of possible images. We would like to take advantage of this freedom in some tractable manner. For example, it may be possible to construct artificial images which are excellent keys for either a specific database, or even for large image domains. Furthermore, our analysis contained the assumption that the query domain was similar to the database. This is not necessarily the case.

Even if we restrict our candidate keys to some random subset of N images, the number of possible subsets of M keys is exponential in M. There is no guarantee that there isn't some elusive set of keys which will prune the database far more than any other set. It may be that heuristics like those traditionally used for NP-complete algorithms may be applicable for key selection. Specifically, the Greedy algorithm could be modified to be optimal over a small number of key changes. We should also examine more clustering algorithms such as *k-means clustering.*

Finally, there has been no published work on the proper number of keys to use for a database of a given size. There is a tradeoff between the elimination power of a set of keys and the execution time required to compare the query to the key set. Some queries may require more keys than other queries for good performance.

8 Conclusions

Of the algorithms tested, Cluster and Greedy clearly gave the best results. The improvement over random key selection was up to a factor of two. As random key selection reduces Best-Match searches to just a few percent of the database, the use of random keys may be perfectly acceptable for smaller databases.

Given n sample database images and m candidate keys, Cluster and Greedy take $O(n^2)$ and $O(mn^2)$ time respectively. Sampling may be necessary when confronted with very large databases. We used sampling in Greedy and not in Cluster, yet Greedy was essentially as good as Cluster in some cases and better than Cluster in the rest.

It is promising that relatively simple algorithms were able to increase performance to the extent shown in this paper. We hope that further work will provide even better performance.

References

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Combinatorial Pattern Matching*, pages 198–212. Springer-Verlag, June 1994.

[2] J. Barros, J. French, W. Martin, P. Kelley, and M. Cannon. Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval. In *IS&T/SPIE - Storage and Retrieval for Still Image and Video Databases*, volume IV, Jan 1996.

[3] A. Berman. A new data structure for fast approximate matching. Technical Report 1994-03-02, Dept. of Computer Science, University of Washington, 1994.

[4] A. P. Berman and L. G. Shapiro. Efficient image retrieval with multiple distance measures. In *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, February 1997.

[5] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *ACM SIGMOD International Conference on Management of Data*,May, 1997, pages 357-368.

[6] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, Apr 1973.

[7] A. Del Bimbo, M. Campanai, and P. Nesi. 3d visual query language for image databases. *Journal of Visual Languages and Computing*, 3, 1992.

[8] A. Del Bimbo, P. Pala, and S. Santini. Visual image retrieval by elastic deformation of object sketches. In *IEEE Symposium on Visual Languages*, pages 216–223, 1994.

[9] D. A. Forsyth, J. Malik, M. M. Fleck, H. Greenspan, T. Leung, S. Belongie, C. Carson, and C. Bregler. Finding pictures of objects in large

collections of images. In *Proceedings of the 2nd International Workshop on Object Representation in Computer Vision*, April 1996.

[10] J. Uhlmann, Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40, 1991, pages 175-179.

[11] L. Wang and D. C. He. Texture classification using texture spectrum. *Pattern Recognition Lett.*, 13, 1990, pages 905-910.

A Analysis of separation with a uniform density function

Given a set of database images S, distance measure d, key k, and uniform density function f on $d(s \in S, k)$. Assume query q is chosen randomly and uniformly from S without replacement. We show that $P(k$ separates $s \in S$ from $q) = (1-T)^2$. We consider three cases, that of $0 \leq d(s,k) < T$, $T \leq d(s,k) \leq 1-T$, and $1-T < d(s,k) \leq 1$. In the first case, k does not separate q from s when $0 \leq q \leq d(s,k) + T$. This occurs with probability $d(s,k) + T$. Summing over the probability of the first case gives us $\sum_{0 \leq zlT} z+T = T^2/2+T^2$. The third case is symmetrical to the first case, again giving us a value of $T^2/2 + T^2$. The middle case occurs with a probability of $1-2T$. In this case, k does not separate q from s when $d(s,k) - T \leq q \leq d(s,k)+T$. This occurs with a probability of $2T$, yielding a joint probability of $(1-2T) * 2T$. Summing up the probabilities of the three cases gives us $2 * (3T^2/2) + 2T - 4T^2 = 2T - T^2$ for the probability that k does not separate q from s. Thus the probability that k *does* separate q from s is $1-(2T-T^2) = 1-2T+T^2 = (1-T)^2$.