Toward Optimal Search of Image Databases

Thomas P. Minka
Matt L. Miller
Ingemar J. Cox
Stephen M. Omohundro
Peter N. Yianilos

NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

A new algorithm and systematic evaluation is presented for searching a database via relevance feedback. The algorithm takes feedback in the form of relative judgments ("item A is more relevant than item B") as opposed to the stronger assumption of categorical relevance judgments ("item A is relevant but item B is not"). It also exploits a learned probabilistic model of human behavior to make better use of the feedback it obtains. The algorithm can be viewed as an extension of indexing schemes like the k-d tree and vantage-point tree to a stochastic setting, hence the name "stochastic search." Experiments with real and simulated users on a database of 4522 images show a significant improvement over other relevance feedback algorithms. In simulations, the amount of feedback required for the new algorithm scales like $log_2 |D|$, where $|D|$ is the size of the database, while a simple query-by-example approach scales like $|D|^a$, where $a < 1$ depends on the structure of the database.

1 Introduction

Searching for digital information, especially images, music, and video, is quickly gaining in importance for business and entertainment. As databases expand, more effective search techniques are needed. A search typically consists of a query followed by repeated relevance feedback, where the user comments on the items which were retrieved. Having a good query language is an important first step, though the meaning of "good" strongly depends on the type of media and user population. This paper focuses on optimizing the relevance feedback phase,

which by contrast can be done in a fairly universal manner: all information about the media and users is encapsulated in a single stochastic model.

The purpose of relevance feedback is to reduce the ambiguity remaining after a query. This ambiguity remains either because of the user's inability to translate or failure to translate their need into the query language. This problem is particularly acute for images and other perceptual media, for which no adequate and universal query language exists. For example, an image database might allow entering keywords of annotated images, providing an example image, drawing a sketch, or asking for particular colors and textures. Each of these has problems, especially when the user is concerned with a quality that requires interpretation of the image.

The first step in developing a relevance feedback algorithm is to determine an unambiguous, quantitative measure of performance, by which one algorithm may be compared to another. Many published results are qualitative, e.g. a sketch-based query might be shown to yield images that intuitively appear to match the sketch. Then relevance feedback is provided, yielding images which presumably are better matches. A more quantitative approach is to count how much feedback is required to achieve a well-defined level of retrieval accuracy.

The desired level of accuracy used in this paper is simple: find a particular target item. This measure is unambiguous, simple and robust. For example, having a large collection of eagle images will make it easy to retrieve random eagle images, but it will remain difficult to find a particular eagle image on the basis of relevance feedback. It is not robust for evaluating query languages, since a very specific query language could make it trivial to find a particular item. A better level of accuracy for that purpose may be to find *several* particular items.

The main advantage of using a single target as the desired level of accuracy is that it admits a detailed analysis. This paper contains the first algorithm that deliberately minimizes the amount of feedback, based on a predictive model of the user. It also represents the first quantitative, domain-independent analysis of relevance feedback algorithms. This is all because of the simple performance criterion. Our analysis can be used as a platform from which to attack more complex search criteria, as outlined in section 5.

The second step is to determine what information the user can still provide, given that the query language has proved inadequate. A common assumption in image databases is that the user should simply keep trying new queries, e.g. repeated query-by-example. This should be regarded as a fall-back option; we can do better. For example, a common assumption document retrieval and in the FourEyes system is that the user can indicate which items are or are not in the same category as the target, without having to specify what that category is (Turtle and Croft, 1992) (Minka and Picard, 1997) (Rui et al, 1997). Inputs are of the form "item A is relevant (i.e. in the category) while item B is not." However, this forces the user to decide on a useful category: one that is not too large or too small and contains the target. This is a drawback in the applications we envision, where the user has little familiarity with the content

Figure 1: The PicHunter user interface.

of the database. Conjuring up a category should be unnecessary for finding a particular target.

This paper makes the more lenient assumption that the user can make relative judgments between items, i.e. "item A is more relevant than item B." This kind of input still can be used effectively without a notion of category or when no retrieved items are in the same category as the target.

Our retrieval system, PicHunter, uses a minimally sufficient interface for this kind of interaction (figure 1). A set of n images is displayed. After selecting zero or more images, the user calls up another set of images by hitting the "GO" button. The selected images are intended to be more like the target than the unselected ones. This cycle is repeated until the target appears. At this point, the search is terminated by selecting the target and hitting the "FOUND" button. The set of user responses to the images displayed totals 2^n possible combinations of selected images and n possible terminations of the search. The interface is intentionally sparse to permit a comprehensive analysis. Additional controls and readouts are suggested in section 5.

The third step is to decide on a representation for the information that the user has provided. The most complete representation for the information received is a probability distribution over possible targets, which is what this paper uses. The initial query, if any, provides the starting condition for this distribution, which may be multi-modal or have other complex structure. Previous algorithms have approximated this distribution with Boolean formulae (Minka and Picard, 1997) or with a refined query (Turtle and Croft, 1992) (Rui et al, 1997). The new algorithm can be specialized to work with these more restrictive representations, if desired.

Once we decide on (1) target testing, (2) relative judgments, and (3) representing the full target distribution, database retrieval reduces to a well-known problem in computer science: comparison searching. The next section discusses this problem and gives a novel solution for the case when comparisons have stochastic outcomes. This generic algorithm, called "stochastic search," can be used for relevance feedback with any kind of database. The average number of inputs required for stochastic search is shown to be significantly smaller than repeated query-by-example in a wide variety of situations. The remaining sections evaluate the performance of stochastic search with real users in the context of searching a database of natural scenes.

2 Multidimensional comparison searching

This section develops a generalization of existing algorithms for comparison searching, so that they can accommodate arbitrary kinds of comparisons, stochastic comparison outcomes, and comparisons made in parallel. These generalizations allow the relevance feedback problem introduced in the last section to be addressed as a comparison searching problem where the user is performing the comparisons.

Comparison searching is the problem of finding T or the closest element to T among a set of elements D, by only making comparisons between an element and T. The set D can be arranged in advance however one pleases, in order to minimize the number of comparisons at search time.

If the elements are one-dimensional and the distance measure is $d(X_1, X_2) = |X_1 - X_2|$, then the optimal solution is well-known: sort D and use a binary search for T. The comparison operation is the query "$X < T$?". The maximum number of comparisons needed for any search is about $\log_2 |D|$, and the average number of comparisons is about $\log_2 |D| - 1$, where $|D|$ is the number of elements in D.

If the elements have higher dimensionality, or the distance measure is different, then the optimal algorithm is not so obvious. In particular, the comparison "$X < T$?" is no longer appropriate. In the relevance feedback scenario, "$d(X_1, T) < d(X_2, T)$?" is a more appropriate operation.

Regardless of the comparison operation, any search algorithm will have the form of a binary tree, where each node is a comparison. This suggests that we might use a standard dynamic programming algorithm for designing binary search trees (Knuth, 1973). Unfortunately, dynamic programming is infeasible unless we can find a way to prune the exponential number of subtrees, and any such technique would have to be distance measure-dependent. See (Omohundro, 1989) for a Huffman-like approach when distance is Euclidean.

The vantage-point tree has recently been proposed as an algorithm for comparison searching with an arbitrary distance measure (Yianilos, 1993). It is a variant of the k-d tree which is based on thresholds of distance measurements, i.e. the query "$d(X, T) < a$?". The search tree is constructed top-down in a greedy fashion, choosing the vantage point X and threshold a which are expected to prune away as much of D as possible. This can be interpreted as optimization with one step of lookahead. Optimizing this criterion over X is difficult when the distance measure is arbitrary, so a Monte Carlo approach was used: sample several random vantage points and choose the one which minimizes the lookahead cost. For generically distributed data, this greedy algorithm has been shown to perform near-perfect (Yianilos, 1993).

The vantage-point tree is easily generalized to comparisons of the form "$d(X_1, T) < d(X_2, T)$?" and the same Monte Carlo approach can be used to find X_1 and X_2 at each step. The resulting tree, when used with Euclidean distance, is a recursive binary subdivision of the representation space. This is

very similar to oblique classification trees (Murthy, 1995), except search time is being minimized, not classification error.

For illustration, this technique was applied to a database $D_{uniform}$ of uniformly-distributed two-dimensional points inside the unit square. The distance measure was Euclidean. Figure 2 plots the empirical average search time for finding an element of $D_{uniform}$ as a function of database size. (The average is over 1000 searches and the database was resampled 10 times. This procedure was used for all of the experiments in this section.) The average time is very close to the optimal time of $\log_2 |D| - 2$. Note that the optimal time is lower than for binary search since two elements are used at each step. Further generalizations of the vantage-point tree are presented in the following subsections.

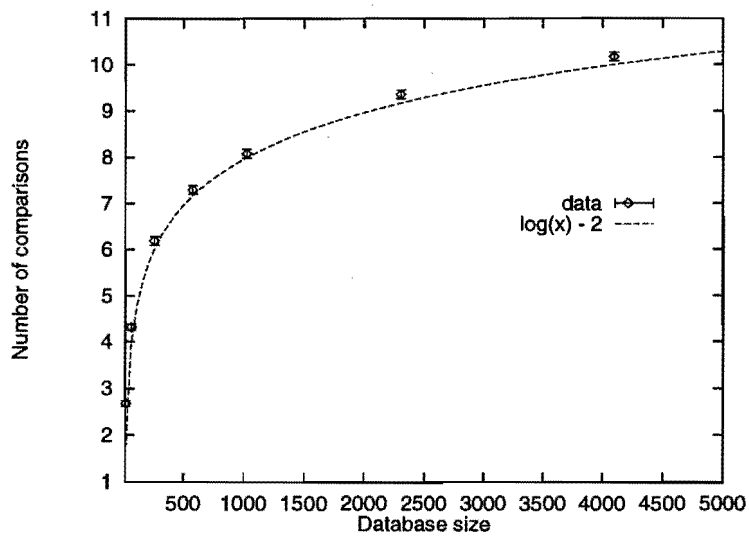

Figure 2: The number of comparisons needed to find an element, for varying database sizes. The database contains uniformly-distributed points inside the unit square.

2.1 Nondeterminism

In the retrieval application we envision, a human will be carrying out the comparisons. This requires that the algorithm be able to cope with an imperfectly modeled, possibly nondeterministic comparison operation. In this more general scenario, the comparison "$d(X_1, T) < d(X_2, T)$?" will return "true" with some probability $p(A = 1|X_1, X_2, T)$. For simplicity, assume that the operation is memoryless, except that making the same comparison again is likely to return the same answer. Section 3 confirms this assumption with real users.

The only change that needs to be made to the above algorithm is the criterion for choosing X_1 and X_2 at each step. In the deterministic case, every comparison completely removes part of D from consideration. That is no longer true: elements are only removed with some probability. In other words, each element X has a probability $p(T = X)$ of being the target T and each comparison operation changes this probability.

The idea is to estimate the number of comparisons left in the search, based on the distribution $p(T = X)$. Call this estimate $C[p(T)]$. Then choose the comparison which minimizes the expected number of future comparisons (the one-step lookahead cost). In other words, the best choice of X_1 and X_2 minimizes

$$(1 - p(T = X_1))(1 - p(T = X_2)) \sum_a C[p(T|A = a)]p(A = a|X_1, X_2) \quad (1)$$

$$p(A = a|X_1, X_2) = \sum_X p(A = a|X_1, X_2, T = X)p(T = X) \quad (2)$$

where $p(T|A = a)$ is the distribution over targets after the answer a (defined below). The $(1 - p)$ factors incorporate the fact that if X_1 or X_2 is the target then there are no further comparisons. (These can be avoided by making "found T = X" a special kind of answer for which $C[p(T|A)] = 0$. This would allow the "found" response to also have a stochastic model. For clarity, this paper assumes "found" to be deterministic and treats this response separately.)

Information theory tells us that

$$C[p(T)] \geq \frac{\text{total information required}}{\text{information from each comparison}}$$

Furthermore, this bound can be achieved by an optimal algorithm. If the denominator is constant, then $C[p(T)]$ is proportional to the entropy of the distribution $p(T = X)$ (which is the total information required), as long as we act optimally. In other words,

$$C[p(T)] \approx -\alpha \sum_X p(T = X) \log p(T = X) \quad (3)$$

for some positive constant α which is irrelevant for the purpose of minimization. This offers an alternative interpretation of minimizing future cost: maximizing immediate information gain. This heuristic has also been used to design classification trees (Murthy, 1995).

Figures 3 and 4 plot the entropy of $p(T = X)$ versus the actual amount of future comparisons needed, averaged over 1000 searches with $|D| = 1024$. Comparison outcomes were deterministic in figure 3 and the relationship is linear. Comparison outcomes were sampled from the $p_{sigmoid}$ model with $\sigma = 0.1$ in figure 4 (this model is defined below). The relationship is linear in two different regimes. This indicates that the amount of information obtained from a comparison is nearly constant, but the constant is different during different parts of the search. A better estimate of $C[p(T)]$ can, of course, be obtained directly from this data, should the linear approximation prove too inaccurate.

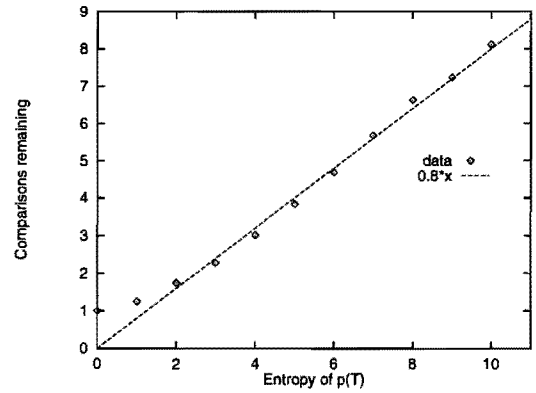

Figure 3: Entropy versus the actual amount of future comparisons needed, averaged over 1000 searches. Comparisons are deterministic. There is a linear relationship, so minimizing entropy can be used to minimize search time.

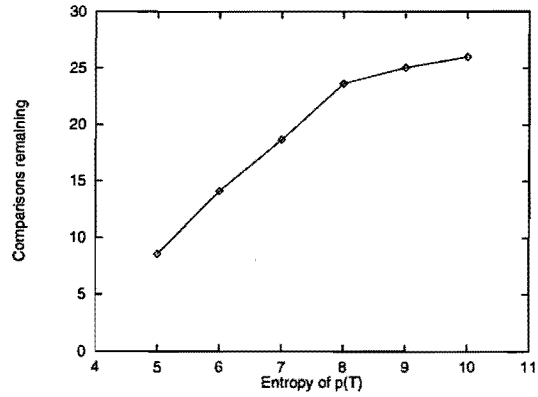

Figure 4: Same as figure 3 except comparisons are nondeterministic. There are two different regimes, but a linear relationship in each.

The entropy criterion generalizes the criterion used in the deterministic case. In the deterministic case, we minimize the number of elements that remain in D as potential targets, until this number reaches one. In the nondeterministic case, we minimize our uncertainty about T, i.e. the entropy of the distribution $p(T = X)$, until this number reaches zero. When $p(T = X)$ is uniform over a subset of elements, the entropy is the logarithm of the number of elements, so the criteria will be the same.

The remaining question is how to update $p(T = X)$ after the outcome of a comparison $d(X_1, T) < d(X_2, T)$. Since this outcome occurs with probability $p(A = 1|X_1, X_2, T)$, Bayes' rule tells us

$$p(T = X|A = 1) = \frac{p(A = 1|X_1, X_2, T = X)p(T = X)}{p(A = 1)} \tag{4}$$

where the denominator is simply a normalization factor that can be computed at run-time. Since the outcomes of different comparisons are assumed independent, the posterior $p(T = X|A = 1)$ can be considered a new prior $p(T = X)$ for the next comparison.

To illustrate this rule, consider the ideal case:

$$p_{ideal}(A = 1|X_1, X_2, T) = \begin{cases} 1 & \text{if } d(X_1, T) < d(X_2, T) \\ 0.5 & \text{if } d(X_1, T) = d(X_2, T) \\ 0 & \text{if } d(X_1, T) > d(X_2, T) \end{cases}$$

If $A = 1$, all elements farther from T than X_1 will get zero probability. The remaining elements will have uniform probability (assuming no ties). Thus we get the usual algorithm for vantage-point trees. Indeed, this is how figure 2 was generated.

Now consider the generalization

$$p_{sigmoid}(A = 1|X_1, X_2, T) = \frac{1}{1 + \exp((d(X_1, T) - d(X_2, T))/\sigma)}$$

When $\sigma = 0$, this is the same as p_{ideal}. When $0 < \sigma < \infty$, there is a smooth transition from probability 1 to probability 0. When $\sigma = \infty$, outcomes are completely random. This formula can be interpreted as p_{ideal} after corrupting the distance measurements with Gaussian noise. The parameter σ can therefore be interpreted as the degree of precision in the distance measurements.

Figure 5 plots the empirical average search time for finding an element of $D_{uniform}$ as a function of database size, when comparison outcomes are sampled from $p_{sigmoid}$. Greedy entropy-minimization scales like $0.77|D|^{0.5}$. This problem is significantly more difficult since the very small distances encountered in large databases cannot be resolved reliably.

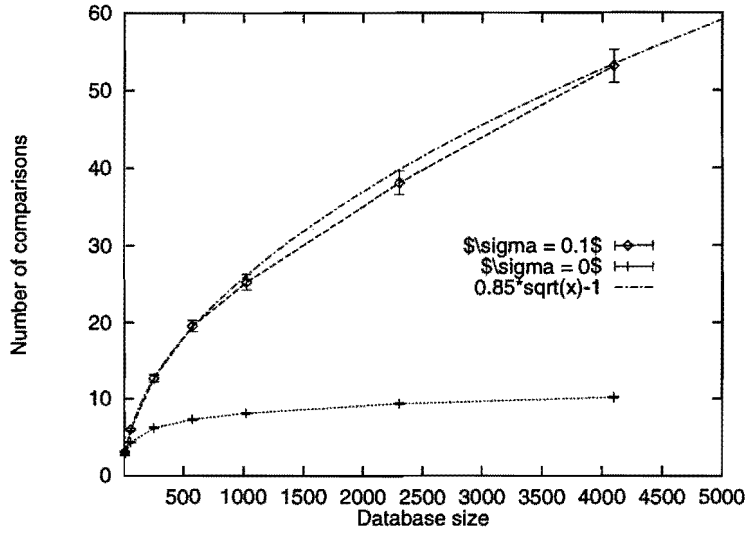

Figure 5: The number of comparisons needed to find an element, for varying database sizes. Comparison outcomes were generated according to $p_{sigmoid}$ with $\sigma = 0.1$. The database contains uniformly-distributed points inside the unit square.

2.2 Simultaneous comparisons

Bringing the problem even closer to reality, suppose multiple comparisons can be made at once. This is readily handled by changing the comparison model.

For example, suppose the user picks the closest element to T from a set of elements $X_1..X_n$. This is equivalent to the $n-1$ individual comparisons "$d(X_a,T) < d(X_i,T)$?" for $i = 1..n$, $i \neq a$. The probability of this outcome is denoted $p(A = a|X_1..X_n, T)$.

If we assume that all of the comparisons are independent, then

$$p_{indep}(A = a|X_1..X_n, T) = \alpha \prod_{i \neq a} p(A = 1|X_a, X_i, T)$$

where α is a normalizing term that depends on $X_1..X_n$ and T. However, independence may not be valid, since the same element X_a participates in all of the comparisons. Independence has so far only been assumed between comparisons with different participants.

Another approach is to generalize the rule for one comparison. For example, the sigmoidal rule $p_{sigmoid}$ can be generalized to a softmax:

$$p_{softmax}(A = a|X_1..X_n, T) = \frac{\exp(-d(X_a,T)/\sigma)}{\sum_i \exp(-d(X_i,T)/\sigma)}$$

which has the natural interpretation of selecting the element with smallest distance to T. In practice, we have found little difference between $p_{softmax}$ and p_{indep} with $p_{sigmoid}$, though the optimal values of σ are different.

Now suppose the user can pick multiple images. If k images are selected, this is equivalent to the $k(n-k)$ individual comparisons "$d(X_a,T) < d(X_i,T)$?" where a is any selected image and i is any unselected image. The probability of this outcome is denoted $p(A = a_1..a_k|X_1..X_n, T)$.

As before, one can assume the selections to be independent:

$$p_{indep}(A = a_1..a_k|X_1..X_n, T) = \alpha \prod_i p(A = a_i|X_{a_i}, X_{unselected}, T)$$

where $X_{unselected} = \{X_1..X_n\} \backslash \{X_{a_1}..X_{a_k}\}$. Or one can compute the softmax between all 2^n possible combinations of elements:

$$p_{softmax}(A = a_1..a_k|X_1..X_n, T) = \frac{\exp(-\sum_j d(X_{a_j},T)/\sigma)}{\sum_{s_1=0}^{1} .. \sum_{s_n=0}^{1} \exp(-\sum_i s_i d(X_i,T))}$$

This seems like more work, but it is not, because the computation of α above also requires summing over all 2^n possible inputs. Again, we can have not found much difference between these two formulae in practice.

2.3 Pseudocode

The "stochastic search" algorithm for comparison searching or relevance feedback works as follows:

1. Find the choices $X_1..X_n$ which minimize

$$(\prod_i (1 - p(T = X_i))) \sum_a C[p(T|A = a)]p(A = a|X_1..X_n) \quad (5)$$

$$p(A = a|X_1..X_n) = \sum_X p(A = a|X_1..X_n, T = X)p(T = X) \quad (6)$$

$$C[p(T)] = -\sum_X p(T = X) \log p(T = X) \quad (7)$$

   If the interface allows multiple selections, summing over all a would be prohibitive. Summing over single selections should be sufficient, though it depends on the comparison model.

2. If one of $X_1..X_n$ is the target, stop.

3. Perform the comparisons, i.e. get the user's response A.

4. Compute $p(A|X_1..X_n, T)$ for all potential targets T. Note that this is zero whenever T is one of the X_i, since we know that none of these are the target. Therefore, these items will never be presented again.

5. Use Bayes' rule to update the distribution $p(T = X)$:

$$p(T = X|A) = \frac{p(A|X_1..X_n, T = X)p(T = X)}{p(A)}$$

6. Iterate back to step 1.

2.4 Comparison to other search methods

Since the minimum possible search time under nondeterminism is unknown, a good way to evaluate stochastic search is to compare it to other plausible methods for choosing $X_1..X_n$:

Most Probable Let $X_1..X_n$ be the n items which maximize $p(T = X)$. Ties are resolved randomly.

Sampling Sample $X_1..X_n$ from the distribution $p(T = X)$.

Query by Example Let $X_1..X_n$ be the n closest items to the winner of the last comparison. This is a favorite approach in systems without relevance feedback. It does not exploit previous comparisons or a model of nondeterminism.

2.5 Deterministic case

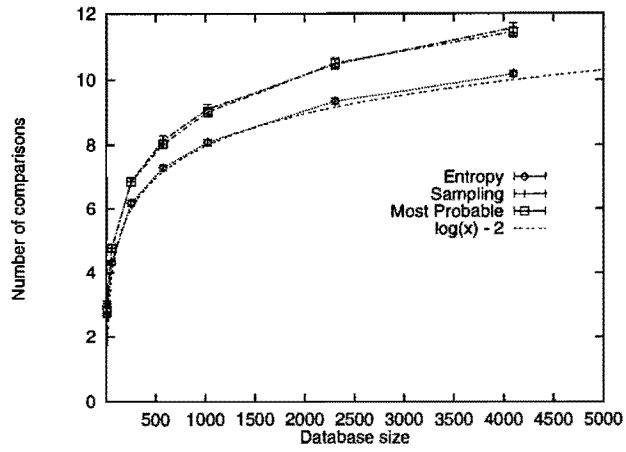

Figure 6: The number of comparisons needed to find an element, for varying database sizes and search strategies. Comparison outcomes were generated according to p_{ideal}. The database contains uniformly-distributed points inside the unit square.

Figures 6 and 7 plot the empirical average search time for finding an element of $D_{uniform}$ as a function of database size, using the four different search strategies. The number of choices n was two. Comparison outcomes were generated by the p_{ideal} model.

In this ideal situation, all remaining targets have the same probability, so Most Probable and Sampling are essentially the same algorithm. Interestingly, both perform as well as entropy-minimization and scale as $\log_2 |D| - 1$. This is probably due to the uniformity of the database. The Query-by-Example method clearly does not exploit comparison information very well; its time scales as $|D|^{0.5}$.

When the dimensionality d of the database is increased, all times remain the same except those for Query-by-Example. Query-by-Example scales as $|D|^{(1/d)}$, probably because the elements become more tightly packed.

The number of choices n can be increased by using p_{indep} with p_{ideal}. The time for stochastic search scales like $\log |D| / \log n$ while Query-by-Example scales like $|D|^{(1/d)}/n$. Thus increasing the number of choices helps Query-by-Example more, and for large enough n there is no difference between the two schemes.

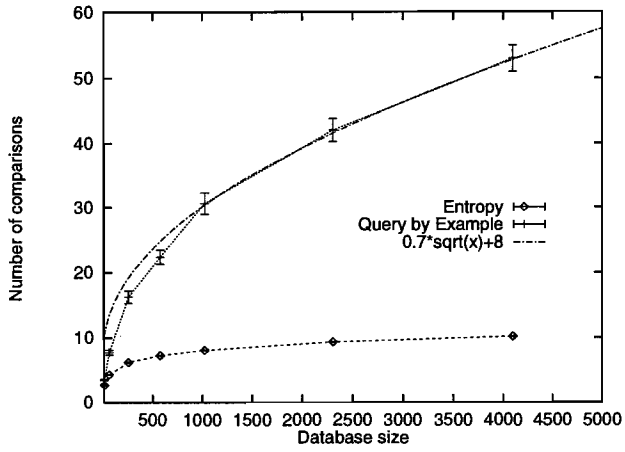

Figure 7: The number of comparisons needed to find an element, for varying database sizes. Comparison outcomes were generated according to p_{ideal}. The database contains uniformly-distributed points inside the unit square.

2.6 Nondeterministic case

Figures 8 and 9 show what happens when comparison outcomes are generated by the $p_{sigmoid}$ model, with $\sigma = 0.1$. Increasing the database size causes the unit square to be sampled more and more finely, while the distance uncertainty threshold σ remains the same. Thus it is much harder to isolate a particular target in a large database than in a small one.

While the Sampling, Most Probable, and Entropy schemes all scale like a square root, they have different leading constants, with Entropy having the smallest.

The Sampling scheme is closest in performance to entropy-minimization. The Most Probable scheme tends to choose items which are close together in feature space—exactly when comparisons are most unreliable. Entropy-minimization, by contrast, automatically chooses comparisons for which answers are reliable. The Most Probable scheme also does not properly exploit very broad and nonuniform distributions, or distributions which are multi-modal. A multi-modal distribution causes this scheme to switch to different parts of the database between iterations, which is disconcerting to the user.

So we find that stochastic search, based on greedy entropy-minimization, is an effective search strategy, always at least as good as the alternatives considered here. It is very general and easy to implement via Monte Carlo optimization. We have also tried using Powell's multidimensional line search algorithm (Press et al, 1992) but with no substantial increase in performance accompanying the significant increase in computation time.

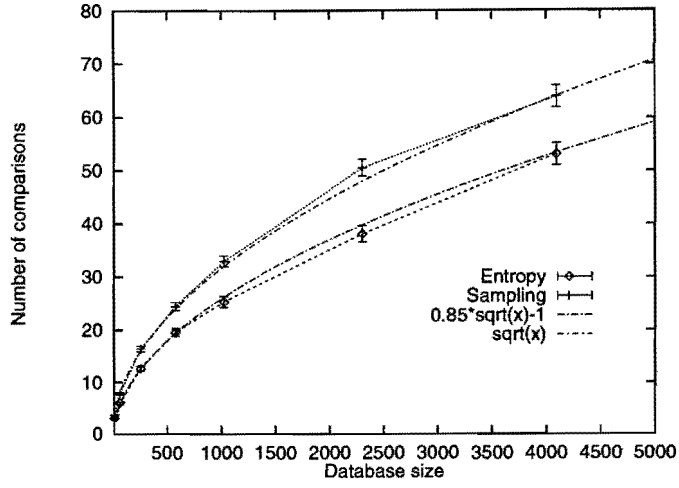

Figure 8: The number of comparisons needed to find an element, for varying database sizes and search strategies. Comparison outcomes were generated according to $p_{sigmoid}$ with $\sigma = 0.1$. The number of choices n was 2. The database contains uniformly-distributed points inside the unit square.

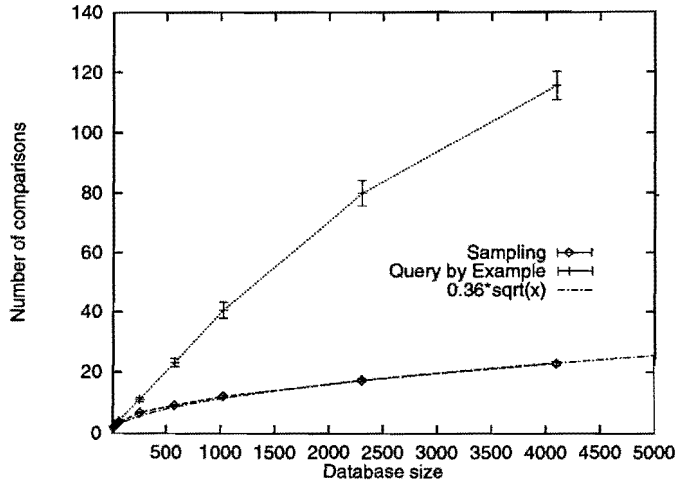

Figure 9: The number of comparisons needed to find an element, for varying database sizes. Comparison outcomes were generated according to $p_{sigmoid}$ with $\sigma = 0.1$. The number of choices n was 4 (the difference is even greater for $n = 2$). The database contains uniformly-distributed points inside the unit square.

3 Developing the User Model

The only part of the stochastic search algorithm which depends on the data and the user population is the probabilistic model for comparison outcomes. This section describes how the parameters of this model can be learned directly from interaction with users.

The database was a commercial collection of stock photographs. We used the 1,500 images from 15 thematic CDs of 100 images each: bald eagles, air shows, Arabian horses, Los Angeles, Israel, Africa, flowers, the arctic, patterns, etc. This assortment makes it easy to find a random image of flowers but rather difficult to find a specific image of flowers, since there are 100 of them. Each image was 128x192 pixels with 24 bits of color.

3.1 Two-image trials

To collect samples from the unknown distribution $p(A = 1|X_1, X_2, T)$, a subject was shown two randomly-selected database images X_1, X_2 beside a randomly-selected target T. The subject was asked to choose the image that was "closest overall" to the target. This yielded 3,681 samples over 9 different subjects.

To model the outcomes, we tried a variety of distance measures suggested by the image database community in combination with the $p_{sigmoid}$ formula (section 2.1). First, the following features were extracted from the images: (*this list is preliminary—Tom*)

HSV-HIST A histogram of the image after conversion to HSV color space and quantization into $4 \times 4 \times 4 = 64$ color bins.

HSV-CORR The color autocorrelogram (Zabih) at distances $1, 3, 5$ and 7. Same preprocessing as HSV-HIST.

RGB-CCV The color-coherence vector (Zabih) of the RGB image after quantization into $4 \times 4 \times 4 = 64$ color bins.

EDGELS The number of "edgels" in the grey-scale version of the image, computed by thresholding the output of a 3×3 Laplacian filter at 20%.

KEYWORDS A list of manually-generated annotations, covering shot type, setting, natural and artificial scene components, as well as specific objects. There were 146 possible keywords.

Note that image annotations are only being used to provide a distance metric between images. This way, the user doesn't need to know what keywords we used or what they mean. We can add new annotations or new computed features at will, without changing the interface. A similar idea has been used in document retrieval, where the user's query is extended, based on feedback, to include terms that he or she may not have known about or terms which have been invented by

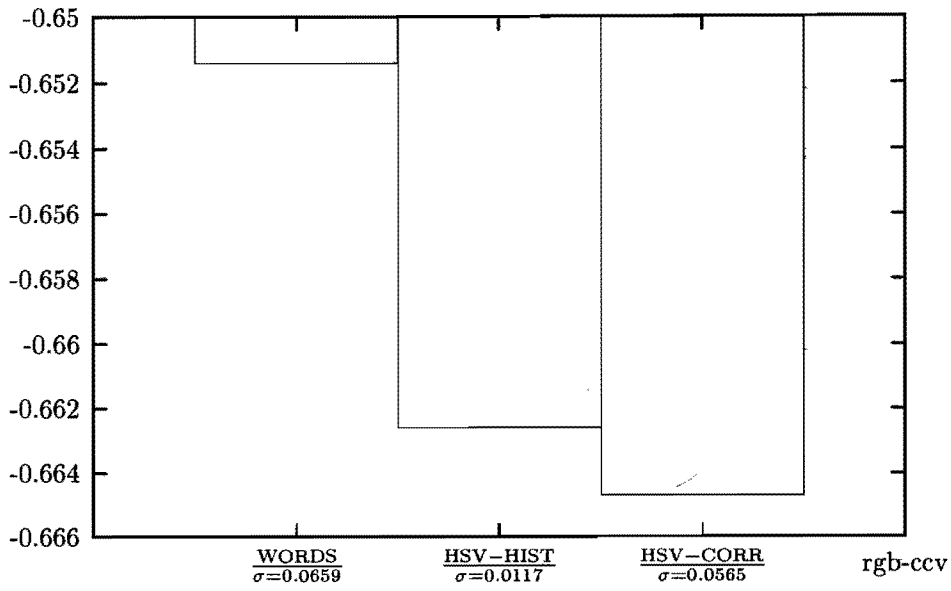

Figure 10: The average log-likelihood of the data collected from nine different subjects, for each of the different image features. The metric was L1 and the probabilities were generated by $p_{sigmoid}$ with the optimal value of σ shown.

the system (Haines and Croft, 1993). Both of these are in contrast to approaches like MediaStreams (Davis, 1995) which rely on a universal iconic query language, containing thousands of icons that the user is expected to understand.

With each feature, there is a choice of distance metric. The L_1 or "Manhattan" distance proved reasonable for all of the features. Using each metric in turn, the average log-likelihood of the data can be computed, as shown in figure 10. The σ parameter was in each case chosen to maximize the average log-likelihood. Finding the maximum is easy with grid search or gradient descent.

The $p_{sigmoid}$ formula is a good model of the data, as shown in figure 11 for the HSV-HIST feature. The graph undergoes no significant changes as the absolute distances $d(X_1, T)$ and $d(X_2, T)$ change, so the difference $d(X_1, T) - d(X_2, T)$ appears to be a sufficient statistic. The optimal value of σ is the same for all of the subjects, so the assumption that all users are identical and all trials are independent turns out to be reasonable.

The next question is whether features can be combined to make a better model. One way is to make a composite distance measure, e.g. by taking a linear combination of the distances according to each feature. Different feature spaces are generally not linearly compatible, but in this case we were able to

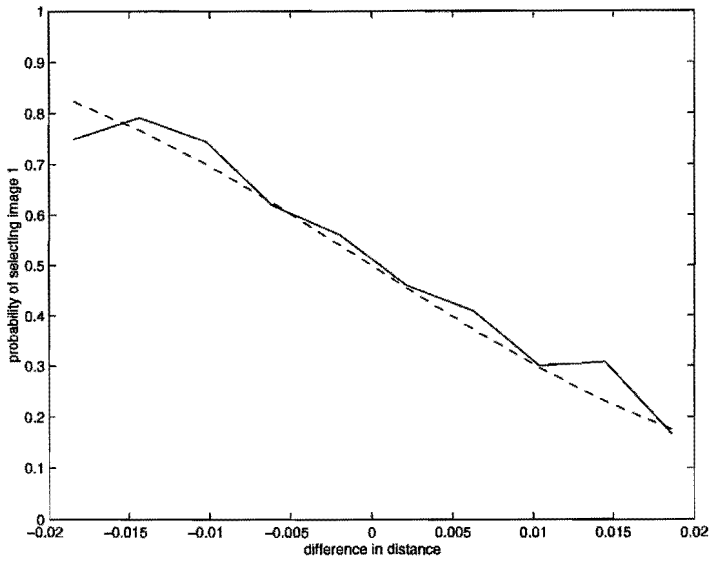

Figure 11: The probability of the subject choosing X_1, given a particular value for $d(X_1, T) - d(X_2, T)$. The output of $p_{sigmoid}$ is overlaid. The feature was HSV-HIST and $\sigma = 0.012$.

substantially improve the log-likelihood to -0.6135 (from the previous best of -0.6514), by choosing the combination weights carefully. Another approach, taken in document retrieval and texture comparison (Picard and Liu), is to linearly combine the *ranks* of the images under the different metrics, as opposed to combining the distances themselves. Unfortunately, combining ranks only reached a log-likelihood of -0.6179.

3.2 Multiple-image trials

Next, subjects performed searches with the `PicHunter` interface, where there were $2^9 + 9 = 521$ possible inputs instead of two. About 6,800 inputs were collected this way.

The best sigmas are not the same as for LTR. But they are the same for different users.

Simulations on Corel go here.

4 Experimental results

This should confirm the simulations done in the last section. Whether PicHunter alone is a useful retrieval system is anyone's guess.

5 Future work

This section presents the remaining open issues in comparison searching, alternative models of the user, and other design choices for the user interface.

5.1 Robustness assessment

The user model for stochastic search is estimated from limited data, which may not be representative of the entire user population. Therefore, it would be useful to predict the search time given that the uncertainty model is itself uncertain. For example, if the algorithm assumes $\sigma = 0.2$, but comparisons are actually being generated according to $\sigma = 0.1$, what is the change in search time? Knowing this function would quantify the algorithm's robustness to the user model. It is easy to perform simulations with these conditions but we have so far been unable to come up with a model for the results.

5.2 Faster response

For large database, complex user model, or large number of choices, entropy minimization can be too slow. In these cases, the Sampling or Most Probable scheme can serve as a replacement. However, it is possible to reduce the computation during search in the following two ways.

When comparison outcomes are deterministic, it is customary to compute and store the entire search tree. Storing the entire tree is prohibitive in the nondeterministic case, so `PicHunter` computes its response at runtime. A hybrid scheme would store a small prefix of the tree, so that Monte Carlo optimization need only be invoked for long searches (exceeding $\log_2 |D|$).

Another approach is to use a search tree computed for the deterministic case in a clever way. This is similar to an idea in the classification tree literature. (The similarity of our modified vantage-point trees with classification trees was pointed out in section 2.) The idea is to pursue both branches of a decision and vote the resulting classifications, weighted by the probability of taking each branch. A similar scheme could work for searching: using a static, conventional vantage-point tree, always explore the branch which has the highest probability of leading to the target. When a path ends without reaching the target, the next most likely path is explored. This scheme will not be as accurate as stochastic search but could be much faster at responding to feedback.

5.3 Feature salience

The user model in `PicHunter` assumes that all of the image features are being used for each comparison. It may be better to exploit the fact that the salience of a feature, such as color or texture, depends on the image being searched for

as well as the images on the screen. If you're looking for a sunset, color may be the determining factor for similarity.

One way to handle this is to ask the user to indicate which image is closest to the target as well as to indicate the notion of closeness. For example, there could be a button for each image feature. Unfortunately, this approach faces the same obstacles confronted by query languages for image data:

- There is often no common vocabulary for image content, especially perceptual content.
- No existing computational model can claim to mimic human perception, even in a simple area like color or texture.

In other words, users will not be able to decide which button to press, and when they do press a button we won't know how to interpret it.

A better approach is to automatically infer what feature was used to make a similarity judgment. For example, the target could have a probability distribution $p(F)$ over features. When comparing an image with the target, $p(F = f)$ is the probability that feature f would be used. The feedback the user has given so far assigns a likelihood to each $p(F)$, making it possible to find the best $p(F)$ for that target and user. Integrating data from many users could provide a prior for $p(F)$ for any target. This prior can be interpreted as the salience of each feature in that image.

5.4 Other kinds of search

As mentioned in section 1, it may sometimes be more appropriate to measure performance by finding several targets, e.g. all images of dogs. This requires a different kind of user feedback and a different knowledge representation. The algorithm would have to represent a distribution over sets of targets, not just targets. Such representations are common in the machine learning literature: Markov random fields (Cressie, 1993), stochastic context-free grammars (Booth, 1973), stochastic Boolean formulae (Cressie, 1993) (Saund, 1995), and various approximations thereof. All of these can be tuned by interacting with users.

A simple kind of feedback is whether a retrieved item is one of the targets (Turtle and Croft, 1992) (Minka and Picard, 1997) (Rui et al, 1997), e.g. contains a dog. The algorithm's job is then to guess what other items might be targets, and present a display which elicits as much information about those other targets as possible. This is exactly the problem of "Active Learning" or "Learning with Queries" studied in the machine learning literature (Freund, 1993). It has roots in optimal experiment design (Federov, 1972). Interestingly, the most common approach in the machine learning literature is maximizing the immediate information gain, in the same way that stochastic search does. We know of no application of Active Learning techniques to database retrieval.

However, there is no need to restrict ourselves to such simple feedback. For example, the user could select the item which is closest to the targets in some average sense. Or the user could pick a different target T at every iteration and choose the closest image to T. This way, there may be no images of dogs shown, but one image may be similar to an image which would have a dog. If there is an image of a dog shown, then the user can give categorical feedback as before, explicitly identifying one of the target images. Allowing the user to switch targets at will would also be useful for open-ended browsing, where the user's goal may change during the course of the search. All of these variants can be handled, as usual, by changing the stochastic model which predicts user inputs.

5.5 Other kinds of feedback

The user can highlight a region instead of selecting an entire image. This is equivalent to having a database composed of the image regions, except that the display can only show entire images. Optimizing the displayed images would be a nightmare.

5.6 Other kinds of readouts

In the other direction, it would be helpful to users to gain more information about the search engine's current "beliefs." This would allow them to better understand the affect of their choices. At the simplest level, the displayed images are presented in the order of estimated probability.

For more detailed information it is useful to know which features caused each image to be displayed. In the simplest variant, each displayed image might have a label such as "texture" or "color." An interesting alternative would be to display a second copy of each image, containing only the pixels that contributed to the selection of that image. For example, an image of an ocean scene might have gained a high probability because the user selected many images which have light blue in their upper regions. In addition to displaying that image, `PicHunter` would display the same image with everything but the sky masked out.

6 Summary

By adopting (1) target search as the performance criterion, (2) relative judgments as the form of feedback, and (3) the full probability distribution over targets as the knowledge representation, `PicHunter` reduces database retrieval via relevance feedback to the problem of comparison searching.

A new algorithm, "stochastic search," exploits a learned model of human behavior to maximize the information it obtains about the user's goal. All

of this information is stored in a probability distribution over targets, which determines the next items to display, and is updated with each user response via Bayes' rule. Consequently, it requires fewer inputs than other relevance feedback schemes, as shown by numerous simulations as well as tests with human users.

A new algorithm, "stochastic search," maintains a posterior probability distribution $p(T = X)$ that item X is the target T. This distribution is used to select the next items to display. The user's selections are then incorporated into the posterior distribution via a probabilistic user model. This model can be learned directly from user interaction by standard statistical methods.

The algorithm requires fewer inputs than other relevance feedback schemes, as shown by numerous simulations as well as tests with human users. In the deterministic case, the average number of inputs required for stochastic search was logarithmic while the average number of inputs required for repeated query-by-example was polynomial in the size of the database. In the nondeterministic case, both scale polynomially, with stochastic search having a consistent advantage.

References

[1] Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, 22:442–450, 1973.

[2] Noel Cressie. *Statistics for Spatial Data.* Wiley, New York, 1993.

[3] Haines D. and Croft W.B. Relevance feedback and inference networks. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2–11, 1993. `http://ciir.cs.umass.edu/info/ciirbiblo.html`.

[4] Marc Davis. Media Streams: An iconic visual language for video representation. In Ronald M. Baecker, Jonathan Grudin, William A. S. Buxton, and Saul Greenberg, editors, *Readings in Human-Computer Interaction: Toward the Year 2000*, pages 854–866. Morgan Kaufmann Publishers, Inc., San Francisco, 2nd edition, 1995. `http://web.interval.com/papers/mediastreams/`.

[5] V. V. Federov. *Theory of Optimal Experiments.* Academic Press, New York, 1972.

[6] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. In *Advances in Neural Information Processing Systems*, Cambridge, MA, 1993. MIT Press. `http://www.research.att.com/ yoav/`.

[7] Jing Huang, S. Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zabih. Image indexing using color correlograms. In *IEEE Computer Vision and Pattern Recognition Conference*, San Juan, Puerto Rico, June 1997. `http://cs.cornell.edu/Info/People/rdz/rdz.html`.

[8] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1973.

[9] T. P. Minka and R. W. Picard. Interactive learning using a "society of models". *Pattern Recognition*, 30(4), 1997.

[10] Kolluru V. S. Murthy. *On Growing Better Decision Trees from Data.* PhD thesis, Johns Hopkins University, 1995. `http://www.cs.jhu.edu/ murthy/thesis/`.

[11] Stephen M. Omohundro. Five balltree construction algorithms. Technical Report TR-89-063, International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA 94704, December 1989.

[12] Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence vectors. In *Fourth ACM Conference on Multimedia*, Boston, Massachusetts, November 1996. `http://cs.cornell.edu/Info/People/rdz/rdz.html`.

[13] R. W. Picard and F. Liu. A new Wold ordering for image similarity. In *Proc. IEEE Conf. on Acoustics, Speech, and Signal Proc.*, pages V–129–V–132, Adelaide, Australia, April 1994.

[14] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 2nd edition, 1992. `http://cfatab.harvard.edu/nr/bookcpdf.html`.

[15] Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Content-based image retrieval with relevance feedback in mars. In *Proc. of IEEE Int. Conf. on Image Processing*, Santa Barbara, CA, October 1997. `http://www.ifp.uiuc.edu/ yrui/html/publication.html`.

[16] Eric Saund. A multiple cause mixture model for unsupervised learning. *Neural Computation*, 7(1), January 1995.

[17] Howard R. Turtle and W. Bruce Croft. A comparison of text retrieval models. *The Computer Journal*, 35(3):279–290, 1992.

[18] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993.