Efficient Filtering with Sketches in the Ferret Toolkit

Qin Lv, William Josephson, Zhe Wang, Moses Charikar and Kai Li
Dept. of Computer Science, Princeton University
Princeton, NJ, USA
{qlv, wkj, zhewang, moses, li}@cs.princeton.edu

ABSTRACT

Ferret is a toolkit for building content-based similarity search systems for feature-rich data types such as audio, video, and digital photos. The key component of this toolkit is a content-based similarity search engine for generic, multi-feature object representations. This paper describes the filtering mechanism used in the Ferret toolkit and experimental results with several datasets. The filtering mechanism uses approximation algorithms to generate a candidate set, and then ranks the objects in the candidate set with a more sophisticated multi-feature distance measure. The paper compared two filtering methods: using segment feature vectors and sketches constructed from segment feature vectors. Our experimental results show that filtering can substantially speedup the search process and reduce memory requirement while maintaining good search quality. To help systems designers choose the filtering parameters, we have developed a rank-based analytical model for the filtering algorithm using sketches. Our experiments show that the model gives conservative and good prediction for different datasets.

Categories and Subject Descriptors

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

General Terms

Algorithms, Measurement, Performance, Design

Keywords

filtering, similarity search, feature-rich data, toolkit, sketch

1. INTRODUCTION

During the past decades, substantial progress has been made on extracting features for similarity search and object recognition from feature-rich data such as audio, image, video, and other sensor datasets. However, not much progress has been made in building efficient content-based search systems for large feature-rich datasets. One of the main reasons for the lack of progress is that searching in high-dimensional spaces is a difficult problem. Unlike in traditional database systems where most searches are on low-dimensional data and can be solved efficiently with a tree-based index data structure (such as B-tree), the dimensionalities of the feature vectors extracted from feature-rich data can be as high as tens or hundreds. To perform k-nearest neighbor (kNN) search efficiently in such high-dimensional spaces has been an open problem since most tree-based indexing algorithms are less efficient than the brute-force approach when the dimensionality is beyond 10 [29]. Some researchers call this problem the "curse of dimensionality".

Recently, the theory community has made some progress on the approximate-nearest-neighbor (ANN) search problem, but proposed solutions have many limitations. The best-known approach is locality-sensitive hashing (LSH) that uses a family of locality-preserving hashing functions to hash a high-dimensional feature vector into buckets in multiple hash tables [13]. To achieve good approximation, however, the proposed approach requires a large number of hash tables (e.g. hundreds) and each table to have as many entries as the number of data objects [11]. Furthermore, LSH approach is limited to a single feature vector and it does not apply directly to multi-feature vector representations and EMD (Earth Mover's Distance) similarity measure in systems such as RBIR (Region-Based Image Retrieval).

A good search mechanism in an efficient content-based search system for feature-rich data should satisfy four requirements. First, it should deliver search results efficiently on large datasets without using much CPU and memory resources. For example, it should be able to search millions of data objects in seconds. Second, it should be able to achieve high search quality by utilizing sophisticated feature extraction methods. For example, it should be able to handle multi-feature vector representations and EMD similarity measure used in a RBIR system. Third, it should be able to search data with multiple modalities effectively. For example, when searching the continuous archived data recorded from multiple medical devices in an intensive care unit, a user should be able to express and search patterns of multiple data sources. Fourth, it should be able to integrate with the keyword-based search engine. For example, users should be able to perform content-based similarity search together with attribute-based search such as time range or annotation-based search.

Princeton's CASS (Content-Aware Search Systems) project investigates how to design efficient content-based search systems that satisfy the four requirements above. We have developed Ferret, a toolkit for constructing content-based similarity search systems for feature-rich datasets and have successfully used the toolkit with five data types including images, audio, personal video, 3D shapes and genomic microarray data. The toolkit includes a basic attribute-based search engine to integrate with the content-based similarity search engine.

The content-based similarity search engine used in Ferret decomposes the similarity search process into two steps: the first step quickly generates a candidate set of similar objects and the second step ranks the data objects in the candidate set. By having the separate two steps, we can use a forgiving approximation method in the filtering process to improve search speed and then apply a sophisticated and perhaps inefficient ranking step to ensure the search quality. A natural way to integrate the content-based similarity search engine with an attribute-based search engine is to filter and rank the results from an attribute-based search. The challenge is to use a high-speed filtering process to generate a small candidate set by filtering out most of the data objects which are dissimilar to the query data object, while retaining most of the similar data objects.

This paper describes the filtering mechanism used in the Ferret toolkit in detail and reports our analytical and experimental results on the filtering technique using sketches. By filtering with the sketches constructed from feature vectors, Ferret can reduce the filtered metadata by up to an order of magnitude and use an extremely fast distance function such as Hamming distance to approximate user-defined segment distance function. Experimental results on several datasets show that the filtering techniques can achieve a factor of 3-10 speedup on search speed while maintaining good search quality. While the brute-force approach can only search about 17,000 images per second, the filtering with sketches method can search more than 100,000 images per second. Also, to help systems designers choose the filtering parameters, we have developed a rank-based analytical model for the filtering algorithm using sketches. Our experiments show that the model gives conservative and good prediction for different datasets.

The rest of the paper is organized as follows. Section 2 gives an overview of the Ferret toolkit. Section 3 describes in detail the filtering algorithm using sketches, and a rank-based analytical model to help systems designers choose filtering parameters. Section 4 reports our performance evaluation of the filtering technique, and how well the analytical model predictions are compared to the experimental results. Section 5 presents the related work. Finally, Section 6 concludes and discusses future work.

2. THE FERRET TOOLKIT

The Ferret toolkit is designed to allow systems builders to construct efficient content-based similarity search systems for various feature-rich data types [19]. This section gives an overview of the software architecture, the similarity search problem and metadata representation in the toolkit.

2.1 Architecture Overview

There are three kinds of components in the Ferret toolkit:

- *Core components* are the key elements of the toolkit that are data type independent. These include the core similarity search engine, an attribute-based search tool, metadata management, and a command-line query interface.

- *Plug-in components* are provided by systems builders to construct similarity search systems for specific data types. There are two plug-in components: segmentation and feature extraction, and distance functions.

- *Customizable components* provide a set of functions that are commonly needed in a similarity search system. These include data acquisition, web interface, and performance evaluation tool.

Users of the toolkit construct search systems by selecting components from the toolkit, supplying a small number of data-type specific routines that adhere to the construction interface, and customizing the user interface, performance, and data acquisition components as necessary. The core components and the data-type specific algorithm implementations are linked into a single, concurrent program, while the data acquisition and user interface modules interact with the search engine either through the function-call level API or remotely via a simple network protocol.

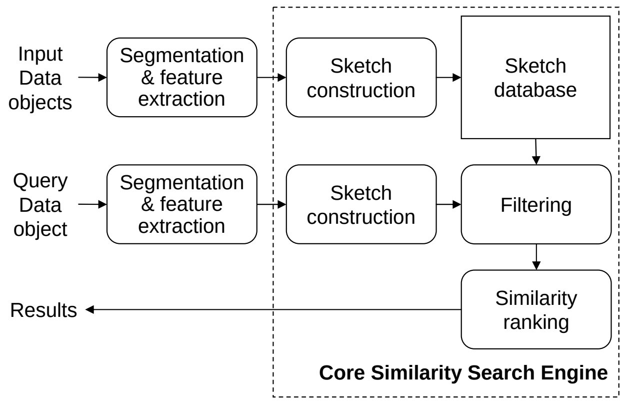

Figure 1: Dataflow in the core similarity search engine of Ferret

Figure 1 shows the dataflow in the core similarity search engine of Ferret. There are two main operations: data input and query processing. As shown in Figure 1, an input data object is passed to a data specific segmentation and feature extraction unit, which is provided by the programmer via the Plug-in interface of the toolkit. This unit will segment the input data object and generate a feature vector for each segment. Each data object is now represented by a set of feature vectors which are then passed to the sketch construction unit. The sketch construction component converts each feature vector into a compact bit vector (sketch). The sketches will then be passed to the sketch database which is managed by the metadata management component.

When a query is presented to the similarity search engine via the similarity search API, the query data is first passed to the same segmentation and feature extraction unit. Similar to processing the input data, the unit will segment the query data into segments and generate a set of feature vectors. The feature vectors will be passed to the sketch construction component to convert them into a set of sketches for the filtering unit and similarity ranking unit.

The filtering unit filters the dataset according to the query data object, either by comparing the sketches against the sketch database or by computing the user-supplied segment distance function directly against all feature vector metadata. The result is a candidate set of data objects. The candidate set may include objects that are not similar to the query object, but by design it misses very few objects that are similar. The similarity ranking component will then be invoked to rank the objects in the candidate set by computing the object distance function plugged in by the user.

2.2 Similarity Search and Multi-Feature Representation

The problem of similarity search refers to searching for objects similar to a query object, *i.e.*, containing similar features. It is also referred to as the *k nearest neighbor (kNN)* problem. Usually, objects are represented by feature vectors and a distance function is used to measure the similarity/dissimilarity between pairs of objects. The complexity of the similarity search problem depends on how complicated the data representation and distance function are. A simple representation is to represent each object by a point in the D-dimensional space. One particular class of distance functions commonly used with such a representation are the ℓ_p norms, where the distance between points $(X_1, \ldots, X_D)$ and $(Y_1, \ldots, Y_D)$ is given by $\left(\Sigma_{i=1}^{D}(X_i - Y_i)^p\right)^{1/p}$.

In many practical settings, feature-rich data is more complex than the simple description above and objects are better represented by a set of feature vectors. Ferret uses a *multi-feature* object representation, which represents an object by multiple segments, each a point in some high-dimensional space with an associated weight:

$$X = \{< X_1, w(X_1) > \ldots, < X_m, w(X_m) >\}$$

where $X_i = (X_{i1}, \ldots, X_{iD})$, $w(X_i)$ is the weight of X_i, and m is variable. Since m may vary from object to object, this representation is flexible and applicable to most feature-rich data types. For example, an image can be decomposed into a set of segments, and each segment can be represented by a D-dimensional feature vector with associated weight to describe its color, shape, coordinates, etc.

With the multi-feature representation, there are now two distance functions. The first is the *segment distance function*, which is the distance between two feature vectors in the D-dimensional space. The commonly used segment distance functions are ℓ_p norms.

The second is the *object distance function*, defined to be the distance between two sets of feature vectors, by matching up the weighted vectors of one object with the weighted vectors of another object in the best possible way.

An example of an object distance function is the Earth Mover's Distance (EMD) [22], which has been used successfully in both image and audio similarity search. EMD calculates the minimal amount of work needed to transform one distribution into another by moving distribution "mass". Given a data object X with m segments, and an data object Y with n segments, EMD(X, Y) is defined as:

$$\text{EMD}(X, Y) = \min \Sigma_i \Sigma_j f_{ij} \cdot d(X_i, Y_j)$$

subject to the following constraints:

$$\begin{array}{rcll} f_{ij} & \geq & 0 & 1 \leq i \leq m, 1 \leq j \leq n \\ \Sigma_{j=1}^{n} f_{ij} & = & w(X_i) & 1 \leq i \leq m \\ \Sigma_{i=1}^{m} f_{ij} & = & w(Y_j) & 1 \leq j \leq n \end{array}$$

where f_{ij} is the extent to which vector X_i is matched to Y_j. An optional implementation of EMD has been included in the Ferret toolkit as the default object distance function. But users can plug in their own object distance function when using the Ferret toolkit.

3. FILTERING

The most general but inefficient method is the brute-force approach that goes through all data objects, computes their distances from the query object, and returns the most similar ones. This approach is extremely inefficient for large datasets using a sophisticated similarity measure such as EMD.

To accelerate the search process, Ferret processes a search in two steps:

- *Filtering* filters out dissimilar objects and generates a candidate set.
- *Similarity Ranking* ranks the objects in the candidate set according to their distances to the query object.

By separating the search process into these two steps, one can use a forgiving approximation method in the filtering process to improve search speed and allow the user to apply a sophisticated and perhaps inefficient ranking step to ensure the search quality. If the candidate set size is small, only a small number of EMD computations are needed in the second step. In addition to speed, the filtering step also provides a natural way to integrate the content-based similarity search engine with an attribute-based search engine.

The goal of filtering is to generate a small candidate set quickly that contains most of the similar data objects. There are two challenging aspects in designing the filtering mechanism. The first is to filter multi-feature data objects whose similarity measure requires complex computations between two sets of feature vectors. The second is to generate a small and high-quality candidate set quickly. The following describes our approaches to address these two issues in Ferret toolkit and shows how to determine proper parameters for filtering.

3.1 Filtering Multi-Feature Objects

The Ferret toolkit is designed to construct content-based search systems for generic feature-rich datasets including multi-feature datasets.

To efficiently filter multi-feature data objects, Ferret's filtering mechanism must avoid complex computations required by the multi-feature object similarity measure or distance function. As discussed in the previous section, Ferret has two distance functions for multi-feature data objects: segment and object distance functions. Typical segment distance functions such as ℓ_1 (Manhattan) and ℓ_2 (Euclidean) are simple and fast, whereas typical object distance functions such as EMD tends to be complex and time consuming.

Our approach is to use segment distance functions to filter out dissimilar data objects. The main rationale of this approach is that the distance between two similar multi-feature objects must have certain number of "match up" segments. Intuitively, if two objects are similar, their "major" segments should be similar. In other words, for an object to be included in the candidate set, it must have at least one segment that matches up well (within a small distance) with a major segment in the query object.

Algorithm 1 shows the pseudo code of the filtering method. The filtering algorithm first selects the r segments with the highest weights of query object Q. This step finds the "major" segments of Q. Next, the filtering algorithm identifies objects in the dataset with at least one segment that is close to one of these high-weight segments. A segment R_j is determined to be close to a high-weight segment Q_i if it is one of the n nearest segments to Q_i.

Algorithm 1 Filtering for multi-feature based object similarity search

```
Input: query object $Q$, $r$, $n$
Output: candidate set $C$

$C = \emptyset$
$Top_r(Q) = r$ largest-weight segments in $Q$

for segment $Q_i \in Top_r(Q)$ do
  $Top_n(Q_i) = Q_i$'s $n$ nearest segments

  for each segment $T_j \in Top_n(Q_i)$ do
    let $T$ be the object that segment $T_j$ belongs to
    $C = C \cup \{T\}$
  end for
end for

return $C$
```

Different methods can be used to find the n nearest segments of a query segment. The simplest way is linear scan, which goes through all segments in the system and finds n segments whose distance are closest to the query segment.

The filtering algorithm can also be combined with indexing techniques to further speed up the query process. Various indexing techniques have been proposed in the literature for the nearest neighbor search problem, but few works well in high-dimensional spaces. A detailed study of filtering with indexing is beyond the scope of this paper. Instead, we focus on filtering techniques that scans through all the segment sketches.

3.2 Filtering with Sketches

Ferret allows users to either filtering with original feature vectors or sketches. Sketches are typically an order-of-magnitude smaller than their original feature vectors and sketch distance can be computed more efficiently than feature vector distance, thus filtering with sketches can speedup the filtering process and reduce metadata size.

The construction unit in the Ferret toolkit constructs a bit vector (sketch) for each high-dimensional feature vector, such that the ℓ_1 distance of two high-dimensional feature vectors can be estimated by computing Hamming distance between sketches, via XOR operations. We give a brief overview of the sketch construction procedure here. Detailed description and proofs can be found in [18].

Let B be the sketch size and H be the XOR block size, both are in bits. To generate a B-bit sketch for a D-dimensional feature vector, we first construct $B \times H$ bits such that the expected distance between any pair of such $B \times H$ bits is proportional to the ℓ_1 distance between the corresponding high dimensional feature vectors. Next, every group of H bits are XORed to produce the final B-bit sketch. The Hamming distances between these sketches are proportional to a transformed version of the ℓ_1 distances between the feature vectors. For feature vectors that are close, their sketch distance is proportional to the original distance. However for feature vectors that are far apart, their sketch distance is proportional to a dampened version of the original distance. Further, this dampening effect increases with H. This dampening (or thresholding) transformation is useful in limiting the effect of large distances.

Sketches are typically an order of magnitude smaller than the original feature vectors, so using filtering with sketches can dramatically reduce the CPU and memory resource requirements. Our experimental results on two large image datasets show that filtering with sketches achieves good search quality with high search speed (a factor of 3-10 speedup), as compared to the brute-force approach.

3.3 Modeling Filtering with Sketches

To design a real system using filtering with sketches, an important design decision is the choice of sketching and filtering parameters, given a targeted dataset size and desired filtering quality. Specifically, the parameters that we need to choose for the filtering and sketching techniques are:

- r: number of query segments per query object
- t: filter ratio – *i.e.* filtered set size is $t \times k$ (where k is the number of similar objects to return)
- B: sketch size in bits
- H: XOR block size in bits for sketching

We now describe a rank-based analytical model for filtering using sketches. This model works for the single-feature scenario (*i.e.*, each object is represented by a single segment)[1] It provides a basis for systems designers to choose appropriate parameter values for a sketch-based filtering similarity search query processor. In particular, for a given dataset size, N, and result set size, k, the model predicts the relationship between recall, filter ratio (t), sketch size (B), and XOR block size (H).

In the following description, let S be a set of N objects, each represented by a D-dimensional feature vector. Let w_i, u_i, l_i be the weight, upper bound and lower bound of the i-th dimension, respectively. Let $T = \sum_i w_i \times (u_i - l_i)$. Given query object q, for object $p \in S$, let $d(q,p)$ be the *feature distance* between q and p, $s(q)$ and $s(p)$ be the sketches of q and p, respectively, and $d_s(q,p)$ the *sketch distance* between q and p. We define the *rank* of p given q to be the number of points in S that precede p when objects are ordered in increasing order of feature distance from q. For a fixed query q, let r_i denote the i-th object in S in this ordering. Similarly, we define the *sketch rank* of p to be

[1] For the multi-feature scenario, we also need to choose the number of query segments r. An analytical model for the multi-feature scenario is much too complicated. Instead, we use experimental results to show how to choose r.

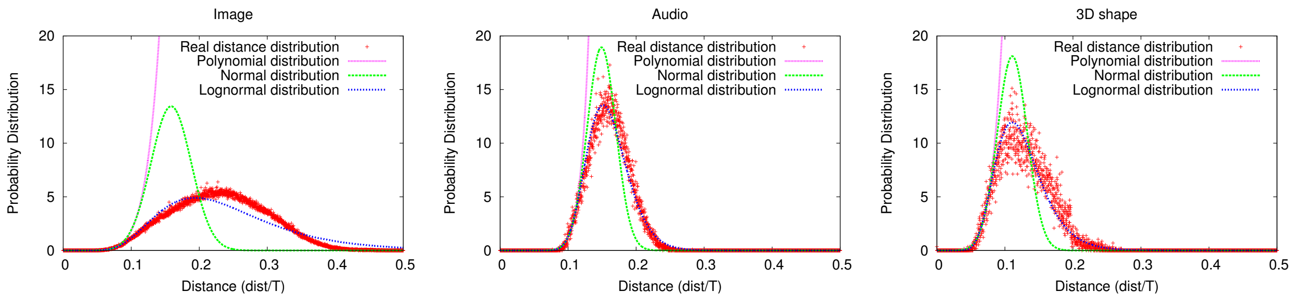

Figure 2: Modeling Feature Distance Distribution.

the number of points in S that precede p when objects are ordered in increasing order of sketch distance to q.

Our analytical model is developed in a series of steps. We describe the results from each step below. For more details, please see our technical report [15].

Feature Distance Distribution. First, we model the feature distance distribution of a dataset. Given a query object q, its feature distance distribution can be calculated as the feature vector distances between q and each object in the dataset. In our study, we consider three common distributions: normal, lognormal, and polynomial. We use the first $2\times k\times t$ data points in the full distance distribution to fit the statistical models, and then use the models to extrapolate to the full set of distances. As shown in Figure 2, lognormal distribution fits the real feature distance distribution best among the three models.

Sketch Distance Distribution. Second, we obtain an expression for the distribution of the sketch distance as a function of the feature distance. For a fixed query point q, consider object $p \in S$ and let $x = d(q,p)/T$. The probability that the two B-bit sketches $s(q)$ and $s(r)$ differ in exactly b bits is given by the binomial distribution:

$$\Pr[d_s(q,r) = b] = p(x,b) = \binom{B}{b} p(x)^b \left(1 - p(x)\right)^{B-b}$$

where $p(x) = \frac{1}{2}\left(1 - (1-2x)^H\right)$.

Rank Distribution. Next, we model the sketch rank distribution of an object for a query q as a function of its feature distance from q. The sketch rank of object p depends on the sketch distance $d_s(q,p)$. Consider the event $d_s(q,p) = b$, consider an object $p' \in S$ such that $d(q,p')/T = x$. Let $P(x,b)$ be the probability that p' is ranked lower (*i.e.* closer to q) than p, we have

$$P(x,b) = \sum_{i=0}^{b-1} p(x,i) + \frac{1}{2} p(x,b)$$

Let $\mathrm{rank}\,(p)$ denote the sketch rank of p, let $f(x)$ be the probability density function for the feature distances. The expected value and variance of $\mathrm{rank}\,(p)$ $(d_s(q,p) = b)$ are

$$\begin{aligned} E_b &= N \int_0^1 P(x,b) f(x) dx \\ Var_b &= N \int_0^1 (P(x,b) - P(x,b)^2) f(x) dx \end{aligned}$$

Given the fact that N is usually at the order of hundreds of thousands, the distribution of $\mathrm{rank}\,(p)$ (for a specific value of $d_s(q,p)$) is approximately normal by the Central Limit Theorem. The normal distribution parameters can be determined by $\mu_b = E_b$ and $\sigma_b = Var_b$. Note that $\Pr[d_s(q,p) = b] = p(x,b)$. The probability that $\mathrm{rank}\,(x)$ is at most M is

$$\Pr[\mathrm{rank}\,(x) \le M] = \sum_{b=0}^{B} p(x,b) \int_0^M f(y;\mu_b,\sigma_b) dy$$

where $f(y;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(y-\mu)^2/2\sigma^2}$,

Search Quality Estimation. Finally, we use the sketch rank distribution to estimate the recall for a given filter ratio value. For a given filter set size M, the expected fraction of the k nearest neighbors being included in the filtered set can be computed as:

$$\mathrm{Recall} = \frac{N}{k} \int_0^{x_0} \Pr[\mathrm{rank}\,(x) \le M] f(x) dx$$

where x_0 can be derived from $k = N \int_0^{x_0} f(x)dx$.

To choose the sketching and filtering parameters for a targeted dataset size and desired filtering quality, one first computes the feature distance distribution using a small sample dataset and obtains the distribution parameters (thus allowing extrapolating to larger dataset sizes). Next, using the rank-based analytical model, one can compute the expected filtering quality (recall) using different B, H, t values. After this, one can simply pick the set of parameter values that delivers the required filtering quality.

4. EXPERIMENTAL RESULTS

In this section, we study the performance of the filtering technique using sketches, and also validate the rank-based analytical model for choosing filtering parameters. We are interested in answering the following questions:

- How effective is the filtering technique using sketches?
- How to choose the filtering parameters? How well can the analytical model help systems designers choose parameters such as sketch size and filter ratio?

4.1 Evaluation Datasets

We have used two evaluation suites. The *multi-feature* evaluation suite is used to evaluate the effectiveness of filtering for multi-feature objects. The *single-feature* evaluation suite is used to validate the analytical model and study how to choose filtering parameters using the analytical model.

	Total	Total	#Seg. / Image	
	#Images	#Seg.	Avg.	Stdev
COREL	59,624	591,610	9.92	4.73
MIXED-WEB	657,379	7,072,108	10.76	4.95

Table 1: Multi-Feature Evaluation Datasets.

	#Objects	Feature Vec. Dimensions
Image	662,317	14
Audio	54,387	192
3D shape	28,775	544

Table 2: Single-Feature Evaluation Datasets.

There are two image datasets in the multi-feature evaluation suite:

- **COREL image dataset** contains about 60,000 JPEG images from the Corel Stock Photo Library. These are general-purpose images grouped by CDs of different themes (e.g. birds, flowers, cars).

- **MIXED-WEB image dataset** contains about 660,000 images crawled from the Internet. These are general-purpose color images in JPEG format.

Table 1 shows the number of segments after using the JSEG [7] image segmentation tool on the two image datasets. Each image is represented by a set of segments, and each segment is represented by its weight and a 14-dimensional feature vector: 9 dimensions for color moments and 5 dimensions for bounding box information (aspect ratio, bounding box size, area ratio, and x, y positions of segment centroid). The image similarity measure is EMD* match [18].

There are three datasets in the single-feature evaluation suite (Table 2):

- **Image dataset** is generated from about 60,000 images in the Corel Stock Photo Library and about 10,000 VARY images [2]. We use JSEG to segment the images into about 660,000 regions. Each region is represented by a 14-dimensional feature vector (see above).

- **Audio dataset** is drawn from the DARPA TIMIT collection [9]. This collection contains 6,300 English sentences spoken by people with different accents. We break the sentences into about 54,000 segments. For each audio segment, a 192-dimensional feature vector (32 windows $\times$ 6 MFCC parameters) is extracted using the Marsyas library [27].

- **3D Shape dataset** contains about 29,000 3D polygonal models gathered from commercial viewpoint models, De Espona Models, Cacheforce models and from the web. Each model is represented by a single 514-dimensional spherical harmonic descriptor (SHD) [16].

In this evaluation suite, each object is represented by a single feature vector. Weighted ℓ_1 distance is used to compute the similarity between pairs of objects.

[2]`http://www-db.stanford.edu/~wangz/image.vary.jpg.tar`

4.2 Evaluation Metrics

To evaluate the performance of a similarity search system, we consider three measures: search quality, search speed, and space requirement. Given a dataset, we randomly pick 100 query objects. For each query object, we use the domain-specific similarity measure to compute its k nearest neighbors. Search quality is measure by *recall*, which is the fraction of the k nearest neighbors that are actually returned in the results. Let A be the set of objects retrieved by the system (the actual answer), and S be the set of objects that are similar to this query object (the ideal answer), *recall* is defined as:

$$recall = |A \cap S| \ / \ |S|$$

Search speed is measured by query time, which is the latency between submitting a query and getting the results. Space usage is measured by the metadata size of the original feature vectors or the sketches.

All experiments are performed on a PC with two 3.00GHz Pentium 4 CPUs. The PC system has 4GB of DRAM and two 250GB 7,200RPM SATA disks. It runs Linux with a 2.6.9 kernel. All results are averaged over 100 queries. Due to the randomness nature of the sketch construction algorithm, each experiment involving sketches is repeated multiple times and the average is reported.

4.3 Effectiveness of Filtering with Sketches

We compare the following three techniques:

- **Bruteforce-Vec** is the baseline approach. It computes the object distances for all objects in the system using segment feature vectors.

- **Filter-Vec** uses filtering to generate a small candidate set and computes object distance only for objects in the candidate set. It uses segment feature vectors to find the nearest segments of a query segment (weighted ell_1 distance).

- **Filter-Sketch** is the same as *Filer-Vec* except that it uses segment sketches to compute segment distance (Hamming distance on bit vectors).

Table 3 compares the search performance of brute-force approach and the filtering methods, using the COREL and MIXED-WEB multi-feature image datasets. As we can see, the filtering methods achieve good search quality and speed up the query time by a factor of 3-10, as compared to the Bruteforce-Vec approach. While the brute-force approach can only process about 17,000 images per second, the filtering with sketches method can search more than 100,000 images per second. Also, the time saving is more significant for the MIXED-WEB dataset (660K images) than for the COREL dataset (60K images). What is more, compared to Filter-Vec, Filter-Sketch achieves better or similar search performance, while using less than half space (22MB vs. 49MB for the COREL dataset, 242MB vs. 517 MB for the MIXED-WEB dataset).

4.4 Choosing Filtering Parameters

To answer the question of how to choose the filtering parameters, we first conduct experiments on the multi-feature datasets to study how many query segments (r) are needed to achieve good performance. Next, we perform experiments

Dataset		Method	Recall	Avg. Query Time (s)	# Images / second	Speedup Factor
COREL		Bruteforce-Vec	1.0	3.60	17k	1.0
	$r = 4$	Filter-Vec	0.88	0.52	115k	6.9
	$n = 1200$	Filter-Sketch	0.90	0.67	89k	5.4
	$r = 3$	Filter-Vec	0.74	0.37	161k	9.7
	$n = 900$	Filter-Sketch	0.86	0.40	149k	9.0
MIXED-WEB		Bruteforce-Vec	1.0	38.7	17k	1.0
	$r = 3$	Filter-Vec	0.85	10.6	62k	3.7
	$n = 6500$	Filter-Sketch	0.83	9.87	67k	3.9
	$r = 2$	Filter-Vec	0.83	5.55	118k	7.0
	$n = 6000$	Filter-Sketch	0.78	5.14	128k	7.5

Table 3: Performance comparison of the brute-force approach and the filtering methods, using the COREL and MIXED-WEB multi-feature image datasets. (B = 128, H = 3)

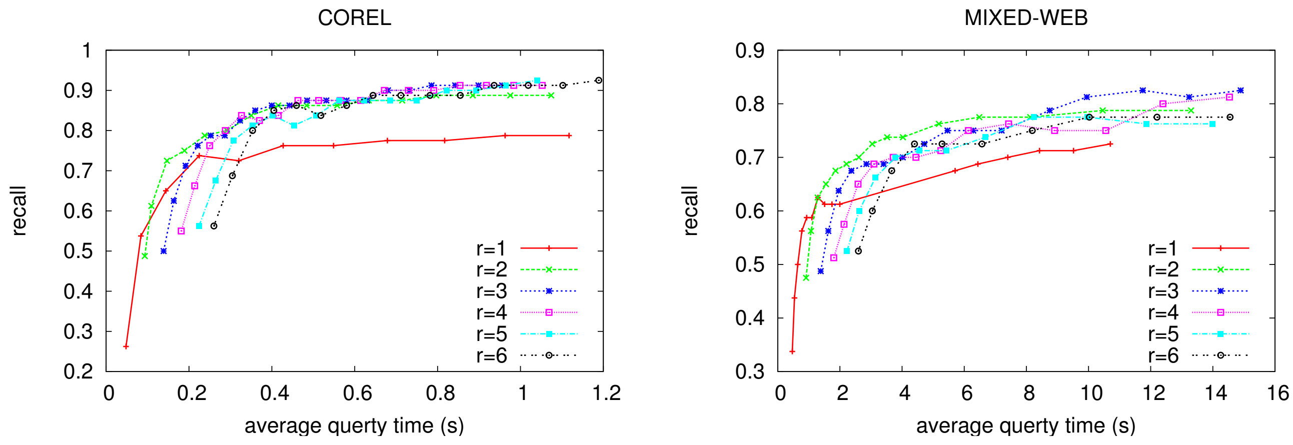

Figure 3: Recall and average query time of the Filter-Sketch method using different number of query segments r, for the COREL and MIXED-WEB multi-feature image datasets. (B=128, H=3)

on the single-feature datasets and study how well the analytical model can help systems designers choose sketch size B, XOR block size H, and filter ratio t. We also report how well the analytical model predicts for large datasets when its parameters are set with a small sample dataset.

Choosing number of query segments r. Figure 3 shows the performance (recall and average query time) of the *Filter-Linear-Sketch* method using different parameter settings. The plots on the first column show the search quality (recall) as a function of the number of results per segment. The plots on the second column show the average query time as a function of the number of results per segment. The plots on the last column show the search quality (recall) as a function of the average query time. A good search system should have high search quality and small search time, so a curve that is closer to the top left corner means better search performance. As we can see, for the COREL image dataset, using 3 or 4 query segments gives better performance, and for the MIXED-WEB image dataset, using 2 or 3 query segments gives better performance.

Choosing sketch size B. Figure 4 shows the filtering quality for different sketch sizes. The red line is the experimental results, and the green line is the prediction by the analytical model using lognormal feature distance distribution. As we can see, the analytical model conservatively predicts the average recall in all cases and the predicted trend is consistent with the experimental results.

Choosing XOR block size H. Figure 5 shows that our analytical model predicts similar H values to the experimental results. For the image dataset, both predicted and experimental results indicate the the best H value is 3. For audio dataset, the model predicts that the best H value is 3 and the experimental results show that the best is 2. For 3D shape data both indicate that the best H value is 4. As before, the model predictions are consistently conservative.

Choosing filter ratio t. Figure 6 shows the results of using the analytical model to estimate the filter quality using different filter ratio for $k = 100$. The results show that the analytical model predicts the filtering quality quite well. Our analytical model predicts the same or similar knee points for all datasets. The knee points are all around a filter ratio of 5, although there is a larger gap between the predicted and experimental results for the 3D shape dataset.

Extrapolating to Larger Dataset Size. When building a real system, it is common not to have the full dataset

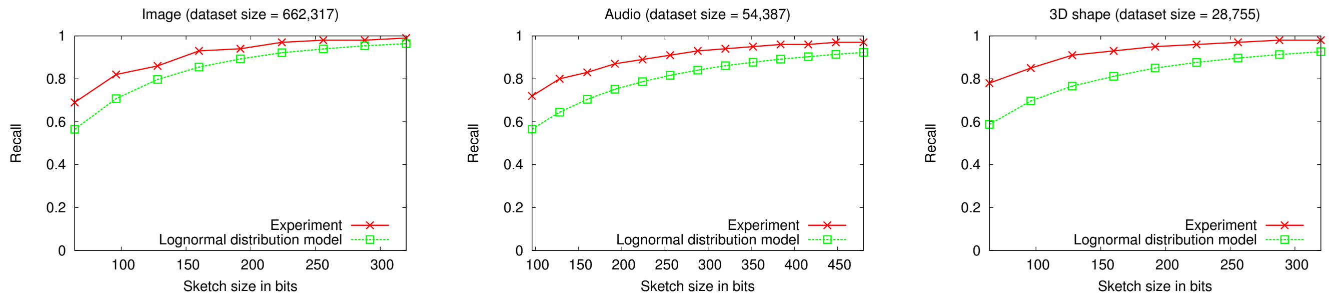

Figure 4: Filter Quality vs. Sketch Size B. (H = 3, k = 100, t = 10)

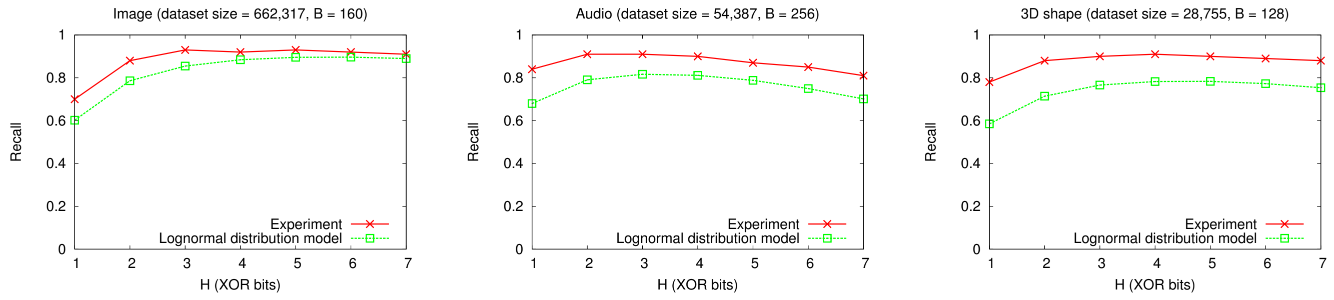

Figure 5: Filter Quality vs XOR Block Size H. (k = 100, t = 10)

available at the initial deployment. It is important to be able to choose system parameters with only a small sample dataset, and have some performance and quality guarantees as the dataset size grows.

In order to validate our model's prediction, we conducted an experiment that simulates dataset growth. For each dataset, we only use one tenth of the total data objects to model the distance distributions and then use the model parameters derived from the sample dataset to predict the filter quality when dataset size grows. The result is compared with the experimental results, where more and more data points in the dataset are included in each experiment to simulate the growing dataset.

Figure 7 shows the filtering quality with different dataset sizes. In each plot, the first data point corresponds to the small sample dataset that we use to derive our model parameters. The results show that the filtering quality degrades gradually as the dataset grows larger. The model can give a good prediction on the degree of quality degradation as the dataset size grows. The prediction works better when the sample dataset size is reasonably large as seen in the image dataset. For other datasets, the degradation prediction is more conservative, but conservative estimates are more acceptable than optimistic in real systems designs.

5. RELATED WORK

Content-based similarity search for feature-rich data has been a popular research topic for over a decade. However, most work focuses on domain-specific data types and feature extraction techniques [28, 25, 23, 4, 14]. The problem of efficient similarity search for general-purpose, large-scale feature-rich datasets has not been solved.

Several recent systems have focused on building a unified index for content in a personal file system. These include Apple's Spotlight [26], MyLifeBits [10], and desktop search tools from Google, Yahoo!, MSN, and others. Search is performed on text-based keywords, attributes, and annotations. None of these systems has addressed how to perform content-based similarity search for noisy, high-dimensional, feature-rich data.

A number of indexing structures (such as K-D tree, R-tree, X-tree) have been devised for nearest neighbor search, as surveyed in [2, 5]. Although these indexing techniques perform well at low dimensions, their performance degrades quickly as the dimensionality increases, so called *curse of dimensionality* [8, 20]. It has been shown that in both theory and practice, if the number of dimensions exceeds around 10, a simple linear scan outperforms all indexing methods that are based on space partitions [29].

More recently, several filter-based indexing techniques have been proposed. Such techniques try to overcome the curse of dimensionality by filtering the vectors (using approximations) so that only a small fraction of them need to be searched when processing a query. For example, VA-file [29] uses a compact, geometric approximation for each vector. By first scanning these smaller approximations, only a small fraction of the vectors need to be visited. Later approaches combine space partitioning and approximation. The A-tree method [24] uses virtual bounding rectangles which contain approximated minimum bounding rectangles and data objects. The AV-tree method [1] approximates data points with prefix paths through a quad-tree-like structure. These filter-based techniques are different from Our filtering algorithm is different in that it is designed for multi-feature based object similarity search, where each object is represented by a set of segment feature vectors, while the filter-based techniques above are designed for single-vector ob-

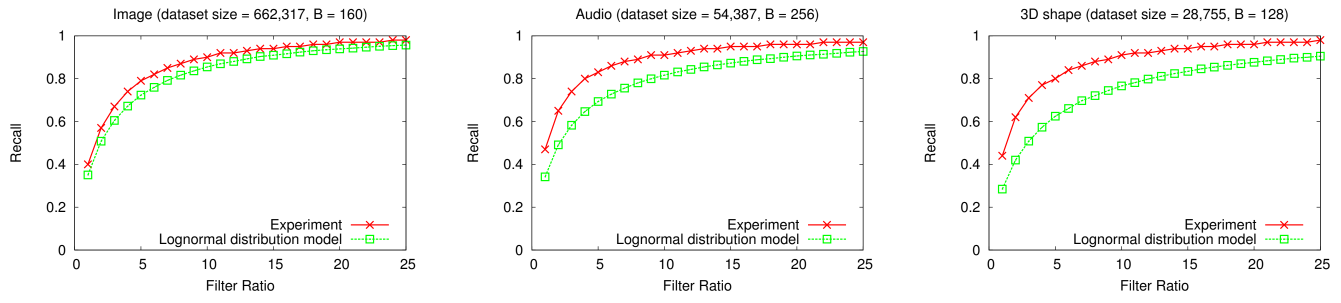

Figure 6: Filter Quality vs Filter Ratio t. (H = 3, k = 100)

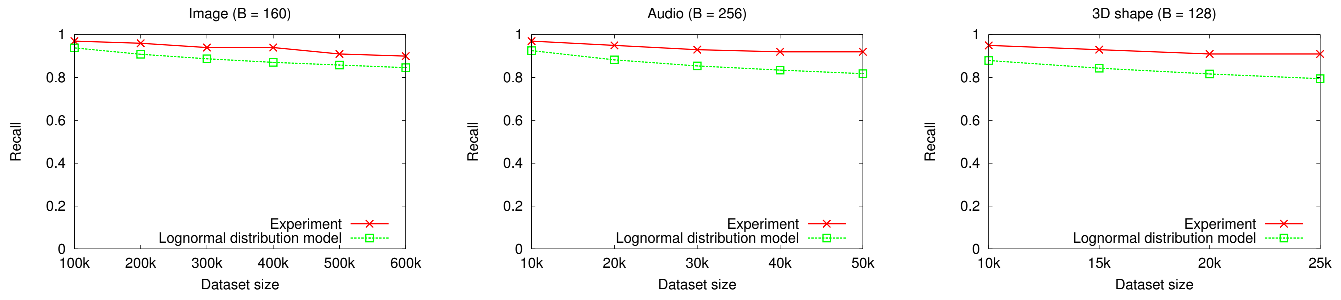

Figure 7: Filter Quality vs Dataset Size (H = 3, k = 100, t = 10)

jects. One other filter-based work is the SnapFind [12] image retrieval system, which allows users to search non-indexed data in a brute force manner. Each query is translated into a customized searchlet that is executed in parallel by processors near the storage devices, enabling early discarsion of the majority of irrelevant images.

To address the "curse of dimensionality" issue, Indyk and Motwani introduced in [13] the notion of locality sensitive hashing (LSH) with which closer objects have a higher chance to be hashed to the same value. This new technique provides fast approximate nearest neighbor search [13, 11, 6]. Recently, Panigrahy proposed a *point-perturbation* based (also called entropy-based) LSH approach [21], which uses perturbed objects in the neighborhood of the query object to check multiple buckets in each hash table, thus reducing the number of hash tables required for good search quality and performance. In [17], Krauthamer and Lee proposed a new indexing structure for nearest neighbor search, called *navigating nets*, whose query time complexity is logarithmic in the number of objects in a dataset. Beygelzimmer *et al.* have proposed another indexing structure, called *cover tree* [3], which improves on the navigating nets technique by using linear search space. These techniques do not directly work with the multi-feature based similarity search for feature-rich data, but may be combined with our filtering technique to further speedup the similarity search process.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we study the filtering with sketches algorithm we have proposed in the Ferret toolkit for efficient content-based similarity search of feature-rich data. Designed for the multi-feature based object similarity search, the filtering algorithm quickly filters out unlikely answers and generates a small candidate set, thus greatly reducing the number of object distance computations needed to answer each query, in effect speed up the search process. Our experimental results show that filtering method achieves good search quality with a factor of 3-10 speedup on the search speed, as compared to the brute-force approach. Also, using sketches for filtering can reduce the space requirement by at least a half while maintaining search quality.

To design a real system using the filtering and sketching techniques, an important design decision is the choice of sketching and filtering parameters, given a targeted dataset size and desired filtering quality. We have developed a rank-based analytical model for the filtering technique using sketches. Our results show that the model gives conservative and good prediction for different datasets. What is more, the parameters of the model can be set with a small sample dataset and the resulting model still gives good predictions for larger datasets of the same type of data.

Currently, our analytical model only works for the single-feature object representation. We are working to extend this model so it can handle multi-feature object representations.

Further optimizations for the filtering algorithm can be investigated. For example, we can consider query segment weights in the filtering process, and obtain more results for query segments with larger weights. Also, the number of segments that we need to examine can be reduced by considering only segments that are big enough (*e.g.*, if an image has several large segments and many small segments, we only need to consider the large segments).

Acknowledgments

This work is sponsored in part by NSF grants CNS-0406415 and CNS-0509447, and by research grants from Google and

Microsoft. Moses Charikar is sponsored by NSF ITR grant CCR-0205594, NSF CAREER award CCR-0237113, and MSPA-MCS award 0528414. William Josephson is sponsored by a National Science Foundation fellowship.

7. REFERENCES

[1] S. Balko, I. Schmitt, and G. Saake. The active vertice method: A performance filtering approach to high-dimensional indexing. *Elsevier Data and Knowledge Engineering (DKE)*, 51(3):369–397, 2004.

[2] C. Böhm, S. Berchtold, and D. A. Keim. Search in high-dimensional spaces - index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[3] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *23rd International Conference on Machine Learning (ICML'06)*, 2006.

[4] A. Cardone, S. K. Gupta, and M. Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *Journal of Computing and Information Science in Engineering*, 3(2):109–118, 2003.

[5] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroqun. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th annual symposium on Computational geometry(SCG)*, pages 253–262, 2004.

[7] Y. Deng and B. S. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(8):800–810, 2001.

[8] D. Dobkin and R. Lipton. Multidimensional search problems. *SIAM J. Computing*, 5:181–186, 1976.

[9] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. DARPA TIMIT acoustic-phonetic continuous speech corpus, 1993.

[10] J. Gemmell, G. Bell, and R. Lueder. Mylifebits: a personal database for everything. *Communications of the ACM*, 49(1):88–95, 2006.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of 25th International Conference on Very Large Data Bases(VLDB)*, pages 518–529, 1999.

[12] L. Huston, R. Sukthankar, D. Hoiem, and J. Zhang. Snapfind: Brute force interactive image retrieval. In *Proceedings of International Conference on Image Processing and Graphics*, 2004.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[14] N. Iyer, S. Jayanti, K. Lou, Y. Kalyanaraman, and K. Ramani. Three dimensional shape searching: State-of-the-art review and future trends. *Computer-Aided Design*, 37(5):509–530, 2005.

[15] W. Josephson, Q. Lv, Z. Wang, M. Charikar, and K. Li. Analysis of filtering for similarity search using sketches. Technical Report TR-760-06, Princeton University, Department of Computer Science, 2006.

[16] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proc. of the Eurographics Symposium on Geometry Processing*, 2003.

[17] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the 15th ACM Symposium on Discrete Algorithms*, pages 798–807, 2004.

[18] Q. Lv, M. Charikar, and K. Li. Image similarity search with compact data structures. In *Proc. of the 13th ACM Conf. on Information and Knowledge Management*, pages 208–217, 2004.

[19] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Ferret: A toolkit for content-based similarity search of feature-rich data. In *Proceedings of ACM Eurosys 2006*, 2006.

[20] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[21] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms(SODA)*, Jan 2006.

[22] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[23] Y. Rui, T. S. Huang, and S.-F. Chang. Image retrieval: Current techniques, promising directions and open issues. *Journal of Visual Communication and Image Representation*, 10(4):39–62, 1999.

[24] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The a-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, pages 516–526, 2000.

[25] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[26] Spotlight: Find anything on your mac instantly. `http://images.apple.com/macosx/pdf/MacOSX_Spotlight_TB.pdf`.

[27] G. Tzanetakis and P. Cook. *MARSYAS: A Framework for Audio Analysis*. Cambridge University Press, 2000.

[28] R. C. Veltkamp and M. Tanase. Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, Utrecht University, Information and Computer Sciences, 2000.

[29] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, pages 194–205, 1998.