Ferret: A Toolkit for Content-Based Similarity Search of Feature-Rich Data

Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li
Department of Computer Science, Princeton University, Princeton, NJ 08540, USA
{*qlv,wkj,zhewang,moses,li*}*@cs.princeton.edu*

ABSTRACT

Building content-based search tools for feature-rich data has been a challenging problem because feature-rich data such as audio recordings, digital images, and sensor data are inherently noisy and high dimensional. Comparing noisy data requires comparisons based on similarity instead of exact matches, and thus searching for noisy data requires similarity search instead of exact search.

The Ferret toolkit is designed to help system builders quickly construct content-based similarity search systems for feature-rich data types. The key component of the toolkit is a content-based similarity search engine for generic, multi-feature object representations. To solve the similarity search problem in high-dimensional spaces, we have developed approximation methods inspired by recent theoretical results on dimension reduction. The search engine constructs sketches from feature vectors as highly compact data structures for matching, filtering and ranking data objects. The toolkit also includes several other components to help system builders address search system infrastructure issues. We have implemented the toolkit and used it to successfully construct content-based similarity search systems for four data types: audio recordings, digital photos, 3D shape models and genomic microarray data.

Categories and Subject Descriptors

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

General Terms

Algorithms, Measurement, Performance, Design

Keywords

similarity search, feature-rich data, toolkit, sketch

1. INTRODUCTION

Digital data volume has been increasing at a phenomenal rate during the past decades [28]. The "Moore's law curve" (doubling every 18 months) no longer refers only to the exponential improvement rate of processor performance, storage density and network bandwidth, but also to the data growth rates of many disciplines [19]. The dominating data types are feature-rich data such as audio, digital photos, videos, and other sensor data. As we are moving into a digital society where all information is digitized and where the world is interconnected by digital means, it is highly desirable for next-generation systems to provide users with abilities to access, search, explore and manage feature-rich data.

A key challenge in implementing a content-based similarity search system for feature-rich data is that such data is noisy and complex. For example, consider two different photographs of an identical scene, or two separate recordings of a person speaking the same sentence. Despite the high degree of similarity between the two images or between the audio recordings, the digital representations are different at the bit level. Comparing noisy, feature-rich data requires matching based on similarity instead of exact match, and thus searching for noisy data requires similarity search instead of exact search. However, similarity search in high-dimensional spaces is notoriously difficult (the so called *curse of dimensionality*). Hence, practical advanced search solutions, such as database tools and search engines (*e.g.* Google), have been limited to searching for exact matches and tend to work only for text documents and text annotations.

This paper presents a toolkit called Ferret that is designed to help system builders quickly construct content-based similarity search systems for various types of feature-rich data. By building a toolkit as the common software infrastructure to solve the core content-based similarity search problems, system implementers can easily add new content-based search capabilities by developing and plugging in data-type specific segmentation and feature extraction modules. This allows implementers to separate their design concerns for specific data types from the core capability of content-based similarity search. With the toolkit, a computer vision engineer can plug in her image segmentation, feature extraction, and distance function modules to construct a content-based similarity search system without worrying about how to deal with metadata management, filtering, indexing and ranking. A genomic researcher can use the toolkit as her research platform to experiment with her new distance functions for finding similar genes. The toolkit for content-based similarity search of feature-rich data will play a similar role to that of the text-based search engines in next-generation operating systems.

A key component in the Ferret toolkit is a general-purpose similarity search engine. To deliver high-quality similarity search results with minimal CPU cycles and memory resources, we have used sketch construction techniques based on dimension-reduction ideas recently developed in the the-

EuroSys'06, April 18–21, 2006, Leuven, Belgium

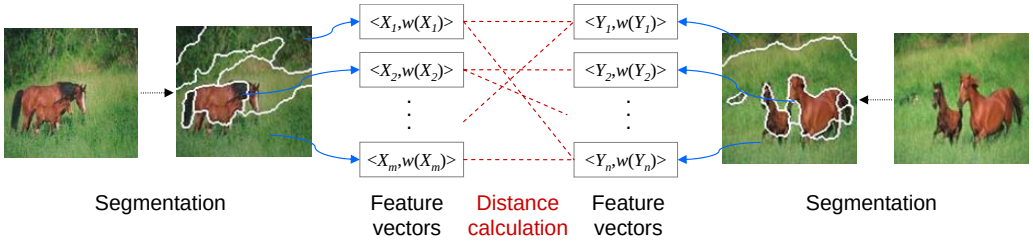

Figure 1: Object representation and distance computation.

ory community. Sketches are tiny data structures that can be used to estimate properties of the original data.

To help system builders quickly construct a content-based similarity search system, the Ferret toolkit provides several other software components, including a metadata manager, an attribute-based search tool, a command-line query interface, a data acquisition system, a web interface, and a performance evaluation tool. The toolkit provides system builders with an API to "plug in" data-type specific segmentation, feature extraction, and distance function modules.

We have designed and implemented the Ferret toolkit, and report user experiences as well as performance evaluation results. Recently, the toolkit has been used to construct content-based similarity search systems for four feature-rich data types including digital images, audio recordings, 3D shape models, and genomic microarray data. Our initial experience has been very positive. Our initial users were able to construct a search system including a web client interface in a few hours. Our experiments show that these search systems can indeed achieve high-quality and high-speed content-based similarity search with modest memory requirements. The sketch technique implemented in the core search engine is quite effective, reducing the storage requirement of feature-vector metadata by an order of magnitude while achieving high speed and high quality.

2. THE SIMILARITY SEARCH PROBLEM

Similarity search refers to searching for objects similar to a query object, *i.e.* containing similar features. Usually, objects are represented by feature vectors and a distance function is used to measure the similarity/dissimilarity between pairs of objects. Given a query object X, the goal is to find all objects Y in the collection such that the distance $d(X, Y)$ is within an allowed range, or to find k objects that are closest to the query object (also referred to as the k *nearest neighbor* problem). The complexity of the similarity search problem depends on how complicated the data representation and distance function are. We designed the Ferret toolkit to support a fairly flexible data representation which we now explain.

A simple representation for objects with D numeric attributes is to represent them by points in 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 $X(X_1, \ldots, X_D)$ and $Y(Y_1, \ldots, Y_D)$ is given by

$$d(X, Y) = \left(\sum_{1 \le i \le D} (X_i - Y_i)^p \right)^{1/p}$$

In some practical settings, feature-rich data is more complex than the simple description above. In many cases of interest, objects are better represented by a set of feature vectors, each a point in some high dimensional space with an associated weight. The general mathematical representation for a feature-rich data object is:

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

where $X_i = (X_{i1}, \ldots, X_{iD})$, $w(X_i)$ is the weight of X_i, and k is variable. Since k may vary from object to object, this representation is flexible and applicable to most feature-rich data types.

Figure 1 shows an example where an image is decomposed into a set of segments. Each segment is represented by a D-dimensional feature vector with associated weight to describe color, shape, area, and coordinates.

With the general representation, there are now two distance functions. The first is the distance between two segments, i.e. the distance between two feature vectors in the D-dimensional space. We refer to it as the *segment distance function*. The commonly used segment distance functions are l_p norms.

The second is the distance between two objects X and Y, which is now quite different from the distance function for the simple representation. Since each object is now represented by a set of vectors and the number of vectors can vary from object to object, we now need to define $d(X, Y)$ between two sets of vectors, by matching up the weighted vectors of X with the weighted vectors of Y in the best possible way. We call such a distance function an *object distance function*. The object distance function is defined using the segment distance function.

An example of an object distance function is the Earth Mover's Distance (EMD) [33], 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". This is a natural distance measure for weighted sets of features and is applicable to image, audio and scientific data. For example, two sound files that exhibit similar segments, but in different order, would be judged similar by the EMD method.

As discussed in the introduction, similarity search in high dimensions is a notoriously difficult problem. Our approach to solving this problem for feature-rich data is to develop approximation techniques. Inspired by recent theoretical results on dimension reduction, we have developed two novel methods for the toolkit. The first is to construct sketches from feature vectors and use sketches to estimate segment distances with simple calculations such as Hamming distance (i.e. how many bits are different between two bit vectors). Since sketches are typically an order of magnitude smaller than the original feature vectors, this approach dramatically reduces the CPU and memory resource requirements.

The second method is to decompose the similarity search process into two steps: filtering and ranking. The goal of the filtering phase is to quickly filter out most objects that are unlikely to be similar to the query object, resulting in a small candidate set. The ranking step performs the more time-consuming but more accurate distance computations for each object in the candidate set and returns the objects that are similar to the query object.

3. TOOLKIT ARCHITECTURE

The main goal of the Ferret toolkit is to allow systems builders to construct efficient content-based similarity search systems for various feature-rich data types.

To accomplish this goal, we set several design requirements:

- The toolkit should include a generic core search engine with an interface that allows domain experts to plug in data-type specific modules.
- The core search engine should achieve high-speed, high-quality searches with minimal CPU and memory resources, even with high-dimensional feature vectors.
- The toolkit infrastructure should solve a set of common implementation issues such as persistence, crash recovery, concurrency, and I/O performance.
- The toolkit should include a set of reusable components that can be customized to build new similarity search systems.

Based on the design requirements, we have designed a toolkit that exports two interfaces. The first is an interface for data-type specific algorithms. This interface allows users to parameterize the toolkit by specifying the implementation of segmentation, feature extraction, and distance functions. As a result, the toolkit provides a platform for domain experts to build new search systems without having to implement the core search engine components from scratch. The ability to parameterize the toolkit by choices for data-type specific algorithms also facilitates algorithm design and implementation since it simplifies the comparison and evaluation of new algorithms on a common infrastructure.

Second, the toolkit exposes a similarity search API to client programs for data acquisition and search. Typical clients of this API include the web interface client, a performance evaluation tool, and automated tools for adding new data to the system. Although we provide default implementations of each of these modules, we expect that users of the toolkit will customize them for their particular applications. For instance, the current web interface to an audio search system that we have built displays file names, text annotations, wave form diagrams, and benchmark performance data, in addition to serving the audio files themselves. Depending on the data type supported by a search system, a richer interface may be necessary.

The toolkit itself provides a core similarity search engine, a simple attribute-based search engine, metadata management, persistence, and crash recovery. These infrastructure components are data type agnostic, but critical to the performance and reliability of a search system. By taking a toolkit approach, we can relieve users of the burden of implementing, maintaining, and improving these generic components.

Figure 2 is a functional block diagram of the toolkit that shows the relationship between the various components. Users

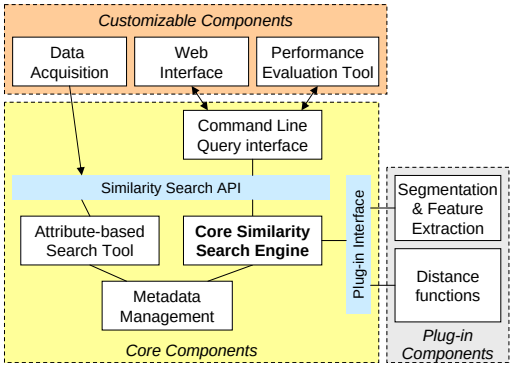

Figure 2: The Ferret Toolkit Architecture

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.

The toolkit consists of the following software components:

- *Core similarity search engine* implements the general purpose similarity search capability for high-dimensional data objects. It constructs highly compact data structures - sketches - for feature vectors and performs filtering and ranking.
- *Attribute-based search engine* implements the search capability by attributes of file objects. Attribute-based search can be used to "bootstrap" similarity search or to further refine an existing query.
- *Metadata management* provides a transaction-protected storage facility for annotations, feature vectors, sketches, and the mapping from data objects to file objects. The metadata store is also responsible for providing persistence across system shutdown and restart.
- *Command-line query interface* processes query commands sent in a command format. This interface supports query processing for web clients and scripting languages.
- *Data acquisition* implements a customizable component that collects new data and inputs data into the similarity search system.
- *Web interface* The toolkit provides a customizable web-based interface for accessing the similarity search engine via the command-line interface.
- *Performance evaluation tool* is a customizable tool that allows domain experts and toolkit authors to run batch queries to evaluate search quality and performance against data-type specific benchmarks. The batch-oriented tools enable an iterative approach to developing data-type specific feature extraction algorithms and distance functions.

To construct a similarity search system, the users of the toolkit need to plug in several modules including segmentation, feature extraction, and distance functions.

4. IMPLEMENTATION DETAILS

This section presents the detailed design and implementation of the components in the Ferret toolkit. There are three kinds of components: core components, plug-in components and customizable components.

4.1 Core Components

Core components are the key elements of the toolkit that are data type independent. There are three components: similarity search engine, attribute-based search tool, and metadata management.

4.1.1 Core Similarity Search Engine

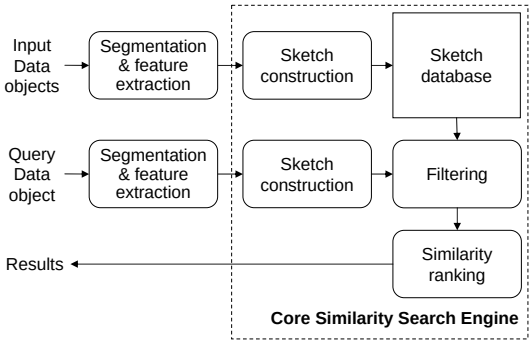

Figure 3: Core Similarity Search Engine

Figure 3 shows the architecture of the core similarity search engine. There are two main operations: data input and query processing. As shown in Figure 3, 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. It will filter out the objects whose distances to the query object are beyond a certain threshold. 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. The ranked results will be returned.

Algorithm 1 Generate $N \times K$ Random (i, t) Pairs

```
input:  N, K, D, min[D], max[D], w[D]
output: p[D], rnd_i[N][K], rnd_t[N][K]
p_i = w_i × (max_i − min_i);      for i = 0, ..., D − 1
normalize p_i   s.t. Σ_{i=0}^{d−1} p_i = 1.0
for (n = 0; n < N; n++) do
  for (k = 0; k < K; k++) do
    pick random number r ∈ [0, 1)
    find i   s.t. Σ_{j=0}^{i−1} p_i <= r < Σ_{j=0}^{i} p_i
    rnd_i[n][k] = i
    pick random number t ∈ [min_i, max_i]
    rnd_t[n][k] = t
  end for
end for
```

Algorithm 2 Convert Feature Vector to N-Bit Vector

```
input:  v[D], N, K, rnd_i[N][K], rnd_t[N][K]
output: b[N]
for (n = 0; n < N; n++) do
  x = 0
  for (k = 0; k < K; k++) do
    i = rnd_i[n][k];      t = rnd_t[n][k]
    y = (v_i < t) ? 0 : 1
    x = x ⊕ y
  end for
  b_n = x
end for
```

Sketch Construction. Sketch construction is the key component to achieve high-speed similarity search and to reduce metadata space requirement. The goal is to convert high-dimensional data into sketches - compact data representations via which one can accurately and efficiently estimate the distance function without using the original high-dimensional data. The challenge is to devise a highly compact sketch which provides accurate estimates.

The sketch construction unit 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. The algorithms were first used for image similarity search [27]. To initialize the sketch construction unit, one needs to specify:

- N: sketch size in bits,
- $min[D]$: min values of the D dimensions,
- $max[D]$: max values of the D dimensions,
- $w[D]$ (optional): the weight of each dimension, and
- K (optional): threshold control whose default value is 1.

When K is greater than 1, the sketch construction unit produces sketches approximating a transformed version of the segment distance (akin to applying a threshold) to reduce the effect of outliers.

We briefly explain the intuition behind the sketch construction procedure and refer the reader to [27] for technical details and proofs. Algorithm 1 shows the initializing process, where $N \times K$ random (i, t) pairs are generated. Then, for every high dimensional feature vector, Algorithm 2 constructs $N \times K$ bits using the (i, t) pairs generated by Algorithm 1. This procedure is designed such that the expected distance between any pair of such $N \times K$ bits is proportional

to the ℓ_1 distance between the corresponding high dimensional feature vectors. Further, every group of K bits are XORed to produce the final N-bit sketch. The Hamming distances between these final bit vectors are proportional to a transformed version of the ℓ_1 distances between the feature vectors. For feature vectors that are close, the distance between the final bit vectors produced is proportional to the original distance. However for feature vectors that are far apart, the distance between the final bit vectors produced is proportional to a dampened version of the original distance. Further this dampening effect increases with K. As explained in [27], the precise values of large distances are not important for the purpose of matching feature vectors. This dampening (or thresholding) transformation is useful in limiting the effect of such large distances.

To speedup similarity search and reduce the storage requirement, users have the option to use compact sketches as the only internal data structures. Our experience shows that the sketch construction can typically reduce the space requirement by an order of magnitude (*i.e.* the bit vectors produced are an order of magnitude smaller than the feature vectors) with minimal impacts on similarity search qualities.

Filtering for Similarity Search. Although it is possible to perform a similarity search by computing the distances from the query object to all data objects in the entire system, this is very time consuming for massive amounts of feature-rich data, due to the fact that most object distance functions are complicated and relatively inefficient to compute. To speedup the query process, the similarity search engine performs a similarity search in two steps: the first step quickly generates a small candidate set of similar objects and the second step ranks the data objects in the candidate set.

The challenge is to find 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. Our filtering approach is to stream through all sketches of the dataset which is typically an order of magnitude smaller than the feature vector metadata and uses an extremely fast distance function such as Hamming distance to approximate user-defined segment distance function. Our experience shows that the filtering technique approximates ℓ_1 and ℓ_2 distance functions quite well in practice.

Since the toolkit uses a generic, multi-feature data object representation, a generic filtering method needs to filter data objects, each of which is represented by a set of weighted feature vectors or a set of weighted sketches. Given a query data object Q, our filtering algorithm selects r segments of Q with the highest weights. Next, we identify objects in the dataset with at least one of their segments close to one of these high weight segments. A segment T_j is determined to be close to a high-weight segment Q_i if it is one of the k nearest (most similar) segments to Q_i and provided its distance from Q_i is within a certain threshold, which is a decreasing function of $w(Q_i)$.

By using the Hamming distance on segment sketches (easily computed by XOR operations), the filtering step can efficiently filter out objects that are unlikely to be similar to the query object, resulting in a much smaller candidate set.

The ranking step then computes the (more accurate) object distance between the query object and each object in the candidate set, thus refining the final answers to the query. In the evaluation section, our performance results show that filtering can greatly speedup query processing, even for datasets with hundreds of thousands of objects.

4.1.2 Attribute-Based Search

Similarity search often needs to work together with attribute or annotation based search. When per-file object attributes or annotations exist, they provide a straightforward way either to "bootstrap" similarity search or to refine a similarity based search. By definition, similarity search requires a "seed" or initial query object. In some cases, this initial object may be readily available, but in other cases the user may wish to identify the seed object on the basis of existing attributes. These attributes may take several forms: generic attributes such as creation time, automatically collected annotations such as GPS coordinates stored with digital photographs, or manual annotations such as notes from a meeting attached to a presentation. What's more, attribute searches can reduce the set of candidates for similarity search and thus improve performance.

Our implementation incorporates a simple keyword attribute search component built on top of Berkeley DB [31]. A separate database table is used to maintain keyword attributes and user-defined annotations for the objects in the dataset. A user can issue an attribute-only query to locate objects of interest and then use those objects as the input to subsequent similarity search queries. The user can also issue queries with both attribute-based search and similarity-based search criteria. When attribute-based criteria are available, the similarity search is performed only on objects that match the query attributes.

4.1.3 Metadata management and crash recovery

The Ferret toolkit also includes metadata management, which provides scalable and consistent storage. From a robustness and performance perspective, it is highly desirable to tolerate power failures or software faults without rebuilding internal metadata.

The metadata management is built on top of Berkeley DB and it keeps the metadata such as feature vectors, sketches, attributes in different tables. All the updates to the metadata associated with the same object are protected by database transactions. This guarantees database consistency in case of system failure. We use the B-tree index provided by Berkeley DB to get efficient keyed access to the metadata.

In the interest of performance, we relax somewhat the ACID guarantees made by Berkeley DB. In particular, we assume that periodic checkpointing of the write-ahead log is sufficient to reduce the window of vulnerability during a crash: updates may not become durable for several seconds or even minutes under high load. However, after a crash, the metadata is guaranteed to be consistent and lost updates can be recomputed by re-acquiring metadata since last checkpoint with substantially less effort than reconstructing the metadata in full.

4.1.4 Command-Line Query Interface

The rationale for a command line query interface is to conveniently support web search interfaces and script languages. When the core components of the toolkit run as a server, we found it very convenient to allow clients to issue queries.

The command-line query interface is designed to process client queries with various parameters including the number of results to return, filter parameters, and attributes as well as data-type specific parameters including adjusted weights

for feature vectors. The command-line query interface allows users to use scripts to quickly experiment with different parameters without restarting the server. The command-line query interface is also used to interface with the web interface and performance evaluation tool.

4.2 Plug-in Components

Plug-in components are provided by system builders to construct a similarity search system for a specific data type. There are two components: the segmentation and feature extraction component and the distance functions component.

4.2.1 Segmentation and Feature Extraction

Segmentation and feature extraction is the key process to "digest" feature-rich data. Since this process is data dependent, the Ferret toolkit must provide a convenient interface such that programmers can "plug-in" new segmentation and feature extraction units easily. In order to work with the general-purpose similarity search engine, the job of a specific segmentation and feature extraction unit is to segment the input data and extract feature vectors. The challenge is to derive a small set of features that characterize the important attributes of the data and to define an effective, general-purpose similarity measure for the feature vectors.

When a new data object is presented to the system, it calls the client-implemented segmentation and feature extraction unit. The interface is defined as following:

```
ObjectT seg_extract_func(const char *filename);
```

The input to this function is the file name of the newly added object. Within this function, the data object is segmented into k segments, and for each segment, a D-dimensional (D is data type specific) feature vector is extracted. There is also a weight w_i associated with each segment, to describe the "importance" of that segment. For instance, in an image of a house and some flowers, the segment corresponding to the house may be considered more important than a small flower segment. The weights for the different segments of an object are normalized so that they add up to 1.

After segmentation and feature extraction, the object X is now represented by a data structure of type ObjectT:

```
ObjectT {
  int        k;          // number of segments
  float*     weights     // segment weights
  FeatureT*  features;   // segment features
}
```

4.2.2 Distance Functions

The Ferret toolkit uses two distance functions for similarity search: *segment distance function* and *object distance function*. The user can either use built-in default functions or define herself.

```
float seg_distance(FeatureT segA, FeatureT segB);
float obj_distance(ObjectT objA, ObjectT objB);
```

When processing a query, *seg_distance()* is used by the filtering unit to quickly filter out unlikely answers, while *obj_distance()* is used by the ranking unit to produce a final ranking of the query results (see details in Section 4.1.1).

The Ferret toolkit has a built-in default object distance function to measure the similarity between two data objects. This distance is based on Earth Mover's Distance (EMD) [33], which has been used successfully in both image and audio similarity searches [33, 23, 18]. EMD is a flexible metric for distributions on high dimensional points, represented by weighted sets of high dimensional vectors. Given two distributions represented by sets of weighted feature vectors and a distance function between pairs of vectors (ground distance function), EMD reflects the minimal amount of work needed to transform one distribution into another by moving distribution "mass" (weights) around. The physical analogy EMD refers to is the process of moving piles of earth spread around one set of locations to another set.

Given a data object X with m segments, and an data object Y with n segments, $\mathrm{EMD}(X, Y)$ is defined as:

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

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.

This is a natural distance measure for weighted sets of feature vectors and is applicable to many data types. For example, two sound files that exhibit similar segments, but in different order, would be judged similar by the EMD method.

A recent study on image similarity search [27] proposes an improved version of EMD which uses a square-root weighting function and thresholding to adjust the importance of different segments. A programmer using the Ferret toolkit can conveniently define her own weighting function and thresholding function.

4.3 Customizable Components

The Ferret toolkit provides a set of customizable components that are commonly used in building a similarity search system, including data acquisition, Web interface and performance evaluation tool.

Data Acquisition. The default data acquisition method is via periodical scan of a designated directory in the file system. Each newly added file in that directory will be imported into the system. This provides a convenient way of continuously importing data if each data object can be stored in an individual data file. For data stored in other formats (*e.g.* in an external database), the data acquisition module will need to be customized to retrieve data from the source properly and input into the system.

Web Interface. The web interface provides users with a simple, yet platform independent way to issue query and present search results. We implemented it by using the Python scripting language to construct a stand-alone web server and connecting it with the Ferret server using the command line interface. Since the basic operation is the same across different application types, we can share the majority of the code for the web interface.

The application-specific presentation part is isolated and can be easily constructed when a new application type is added. The typical use of the web interface is to use attribute-based query to bootstrap search process and to issue similarity-based queries to find relevant content. In our experiments, we find it very useful to be able to visualize search results.

Performance Evaluation Tool. One important common task for building a similarity search engine is to determine

parameters and to experiment and evaluate the system performance and search quality. The Ferret toolkit contains a performance evaluation tool which can drive various performance tests. The input we take is a formatted benchmark file containing the performance benchmark suite which describes the ground truth for similarity search result. Once the benchmark file is given, we are able to drive the test and provide statistics like average precision and time spent for the query.

5. USING THE FERRET TOOLKIT

We have built similarity search systems with the Ferret toolkit for four different data types: image data, audio data, 3D shape data, and genomic microarray data. The first three systems were built by the authors of this paper, and the last one was built by an external research group.

To build a similarity search system using the Ferret toolkit, a programmer needs to provide the following:

- A segmentation and feature extraction module.
- A segment distance function to compute distance between two segments, and an object distance function between two data objects.
- Parameters for sketching, filtering and ranking, including sketch size, number of query segments to use in filtering, number of filtered candidates to get for each query segment, and number of similar objects to return after ranking.

The performance evaluation tool and the web interface in the toolkit can be used for tuning parameters.

The following describes each use case including its segmentation and feature extraction module, distance functions, and possible user applications. We will comment on our initial experience integrating these data types at the end of this section, and provide some insights for programmers who may be interested in building their own similarity search system with the Ferret toolkit.

5.1 Image Data

For image data, we chose the Region-Based Image Retrieval (RBIR) approach as the base method to construct an content-based image similarity search system. RBIR methods segment an image into multiple homogeneous regions based on color and texture. A region is then represented using a combination of color, texture, shape, and spatial information. It has been shown that RBIR methods provide better search quality than the traditional Content-Based Image Retrieval (CBIR) methods which uses global features such as the color and texture of the whole image [40].

Segmentation. We chose to use the publicly available JSEG segmentation tool [12] to segment images into segments. Figure 1 shows two images segmented using the segmentation tool. The segmentation tool reads in an image and outputs a matrix mapping each pixel to one of the segments. A user can easily replace this tool with a new image segmentation tool she has.

Feature Extraction. The feature extraction method is proposed in our recent study on image similarity search [27]. In this method, each image segment is represented by a 14-dimensional feature vector: 9 dimensions for color moments and 5 dimensions for bounding box information. Color moments is a compact representation of color distribution and it has been shown in [29] that the performance of color moments is only slightly worse than high-dimensional color histograms. The bounding box is the minimum rectangle covering a segment. We use 5 features to characterize it: aspect ratio (width/height), bounding box size, area ratio (segment size/bounding box size), and segment centroids.

The weight of each segment is proportional to the square root of that segment's size. The weights are normalized such that the sum of all region weights is 1.

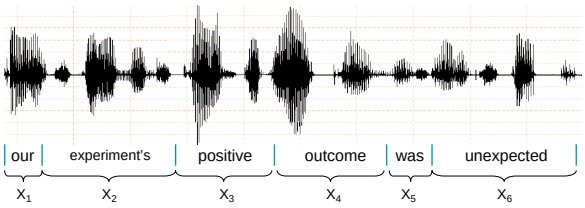

Figure 4: Audio Segmentation

Distance Functions. The segment distance function we use is weighted ℓ_1 distance on the 14-dimensional feature vectors. We use the built-in sketching functionality of the toolkit to filter images before ranking with the object distance function.

The object distance function is a thresholded EMD function. Segment distances are thresholded before EMD computation to reduce the impact of segment "outliers" on the overall object distance.

5.2 Audio Data

Although speech recognition has been well-studied in the audio processing community, little work has been done on generic audio similarity search. Some previous work has considered the problem of music genre classification [42], but there has been relatively little work on speaker-independent similarity search for speech data. One possible approach using existing tools is to use speaker-independent speech recognition to obtain a text transcript of the speech audio data, and use the transcript as the basis for similarity search. However, due to the generally low accuracy of large vocabulary speaker-independent recognition systems (60% at best under optimal acoustical conditions) [37], and given that our goal is to rapidly search audio data of all types, we have opted to take a non-speech, non-text approach to construct a similarity search system for audio speech recording data.

Segmentation. We segment audio data in two steps. The first is to create data objects from a piece of audio data by using an utterance-level segmenter that finds pauses indicating the boundaries between statements. These boundaries are identified by considering the audio signal over 20ms windows and computing both the number of zero crossings and the root mean square (RMS) energy of the signal. The presence of ten or more windows with RMS energy below a certain threshold is taken to indicate an utterance boundary unless there are a large number of zero crossings, which typically indicate the presence of unvoiced consonants [32]. The next step is to break utterances (data objects) into smaller segments that can be compared with one another. For the evaluation of audio data, we use the human marked word boundary in the TIMIT database [16] to segment the sentence.

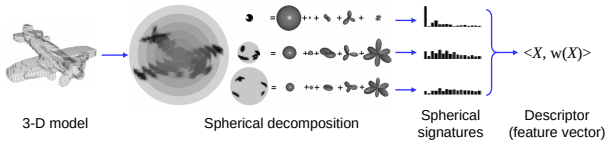

Figure 5: 3D Shape Model Segmentation

Feature Extraction. For each audio segment, we use the Marsyas library [41] to extract feature vectors. We use a 512 sample sliding window with variable stride to obtain 32 windows for each segment. We then extract the first six MFCC parameters from each window to obtain a 192 dimensional feature vector for each segment. The weight of each segment is proportional to the length of that segment, and normalized so that the sum of all segment weights in one data object equals to one.

Distance Functions. We have chosen to use ℓ_1 distance as the segment distance function and use the built-in sketch construction and filtering mechanisms to filter out most of the sentences.

EMD is used as the object distance function. Using EMD has the advantage that it does not respect order and hence allows us to find similar sentences with the same words spoken in a different order.

5.3 3D Shape Models

Shape-based matching and retrieval from databases of 3D polygonal models is useful in computer vision, mechanical CAD, archaeology, molecular biology, paleontology, medicine, computer graphics, and several other fields. Various shape matching algorithms for 3D models have been proposed in the literature [25]. Usually, complicated shape descriptors (with hundreds of or even thousands of dimensions) are computed and a distance function such as ℓ_2 is used to determine the similarity between two models.

To construct a 3-D shape similarity search system, we collaborate with the Princeton Shape Retrieval and Analysis group. One of the authors modified their segmentation, feature extraction, and distance function modules based on the Ferret toolkit's Plug-in API and was able to construct a search system in a few hours.

Segmentation. Each 3D model is represented by an Object File Format file with the polygonal surface geometry of the model. Each model is first normalized, then placed on a $64 \times 64 \times 64$ axial grid. 32 spheres of different diameters are used to decompose the model, and there is one spherical descriptor representing the intersection of the voxel grid with each concentric spherical shell. See Figure 5.

Feature Extraction. The shape descriptor used in our system is called Spherical Harmonic Descriptor (SHD) [25], which is rotation invariant. Values within each of the 32 spherical shells obtained in the segmentation phase are scaled by the square-root of the corresponding area and represented by their spherical harmonic coefficients up to order 16.

When comparing two 3D models (represented by two sets of spherical descriptors), we only compare spherical descriptors corresponding to the same spherical shell at the same scale. Thus, we can "concatenate" all the spherical descriptors of a 3D model in a predefined order, resulting in a $32 \times 17 = 544$-dimensional shape descriptor for each 3D model.

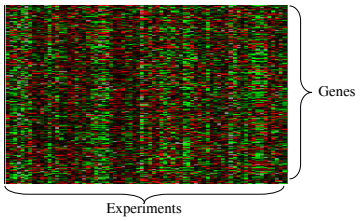

Figure 6: Gene Expression Microarray Data

Distance Function. Since each object has only one feature vector. Only one distance function is needed; the segment distance function is the object function. Although the original search system by the Princeton Shape Retrieval and Analysis Group used ℓ_2 distance, we chose to use ℓ_1 distance and use the sketching mechanism to estimate the distance between two shape descriptors.

5.4 Genomic Data

Research group at the Princeton Bioinformatics and Functional Genomics Lab used the Ferret toolkit to construct search tools for their data analysis. Since searching genes with similar functionality is crucial in the study of gene functions, they use the toolkit to construct search tools to analyze gene expression microarray data in the context of gene function prediction. Specifically, they want to evaluate which types of features and feature selection techniques are most effective for identifying similarly expressed (behaving) sets of genes.

Segmentation. The genomic data are represented as matrices of real valued or binary measurements, where a value in row i and column j is the expression level of gene i in experiment j (in case of microarrays) or the representation of whether genes i and j interact. Each data object is the expression of a gene. Figure 6 shows some example microarray expression data, in which different colors mean different expression levels. Here, segmentation only requires segmenting the big matrix row by row to get the information for each gene.

Feature Extraction. The expression data for each gene is used directly as the feature vector (a row in the microarray genomic dataset). For a given dataset, each gene has only one feature vector.

Distance Function. Since each gene only has one feature vector (segment), the same distance function is used for both segment distance and object distance. The research group has been experimenting with three different distance functions to compute the distance between two genes, including Pearson correlation, Spearman correlation, and ℓ_1 distance.

5.5 Summary

Our initial experience with using the Ferret toolkit has been positive. The general-purpose design of our toolkit has made it convenient to build similarity search systems for different types of data. Given domain-specific segmentation and feature extraction routines and appropriate distance functions, it is straightforward to plug in modules and to set parameters for a new type of data.

To construct the similarity search system for 3D shape data, one of the authors was able to complete a working system with a web interface up and running in less than

two hours. This process included writing wrapper code for the Plug-in Interface of the toolkit and customizing the web interface code to work with 3D models. The main reason for the fast system construction is that we reused the segmentation, feature extraction, and distance function codes written by our collaborators. The performance evaluation section shows that the resulting 3D shape similarity search engine provides similar quality search results as existing systems with metadata only 1/30 of that used by previously reported systems [36].

The researcher at the Princeton Bioinformatics and Functional Genomics Lab reported that the Ferret toolkit is fairly easy to use and that the system is fast and efficient.

6. PERFORMANCE EVALUATION

As described in the previous section, it is possible and fairly easy to build content-based similarity search systems using the Ferret toolkit. This section reports our investigation on performance issues of the Ferret toolkit, namely:

- Can the systems built with Ferret toolkit achieve high-quality similarity search results at a high speed?
- How small can the sketches be as the metadata of the similarity search systems?
- How much benefit can we get by using the sketching and filtering techniques?

To answer these questions, we have conducted experiments with three of the systems built with Ferret: image search, audio search and 3D shape model search.

6.1 Benchmarks

We have used two benchmark suites in our evaluation: a search-quality benchmark suite and a search-speed benchmark suite. Each of the benchmarks in the search-quality suite has a number of predefined similarity sets of unordered data objects as the gold standard. Ideally, using any object in a similarity set as the query item should retrieve the other objects in the similarity set as highly ranked search results. To test search quality, we use one of the data objects in a similarity set as the query data object in a similarity search. We will then use the search-quality metrics mentioned below to evaluate the quality of this particular search.

There are three benchmark datasets in the search-quality benchmark suite:

- **VARY Image Benchmark** is an image collection with about 10,000 general-purpose images [21]. A group of researchers manually defined 32 similarity image sets for this collection [38]. These sets are in different image categories. Our experiments use these 32 similarity image sets for search quality evaluation.
- **TIMIT Audio Benchmark** is an audio collection from the DARPA TIMIT speech database that contains 6,300 English sentences spoken by 630 speakers [16]. We define 450 speech similarity sets, where each set contains 7 utterances of the same sentence spoken by 7 different people. Our experiments use the 450 speech similarity sets for search quality evaluation.
- **PSB Shape Benchmark[36]** is a collection of 1,814 3D shape models. The original benchmark designates 907 models as its training set and the rest as test set. The training set is classified into 90 classes, and the test set has 92 classes. Our experiments use the 92 classes of the test set for the search quality evaluation, and each of the 907 models in the test set is used as a query.

There are three benchmark datasets in the search-speed benchmark suite:

- **Mixed Image Dataset** contains about 600,000 images, mostly crawled from the Web. This dataset also includes about 60,000 images from the Corel image collection.
- **TIMIT Audio Dataset** is the same TIMIT audio benchmark collection. We reuse it to do the search speed test.
- **Mixed Shape Dataset** [5] has about 40,000 3D polygonal models, including about 36,000 free models downloaded from the Web, about 2,000 commercial viewpoint models, about 1,000 commercial De Espona Models, and about 2,000 commercial Cacheforce models.

6.2 Evaluation Metrics

We have used three measures as the metrics to answer the questions above: speed, space requirement, and search quality. To measure search speed, we use the average running time of all the queries on our benchmark datasets. To measure the space requirement, we use different sketch sizes and show the corresponding search quality and search speed. To measure similarity search quality, we use three commonly used search-quality metrics: first-tier, second-tier and average precision:

- **First-tier** is the percentage of data objects in the query similarity set that appear within the top k search results, where k depends on the size of the query similarity set. Specifically, for a query similarity set Q, $k = |Q| - 1$. The first-tier statistic indicates the recall for the smallest k that could include 100% of the data objects in the query similarity set. Suppose $Q = \{q_1, q_2, q_3\}$ and the query is q_1. If the top 2 search results are r_1 and q_2, the first-tier score is 50%.
- **Second-tier** is similar to the first tier except that $k = 2 \cdot (|Q| - 1)$. The second-tier is less stringent since k is twice as large. The ideal result is still 100% and higher values mean better similarity search results. Suppose $Q = \{q_1, q_2, q_3\}$ and the query is q_1. If the top 4 search results are r_1, q_2, q_3, and r_4, the second-tier score is 100%.
- **Average precision** considers where data objects of a query similarity set appear in the search results. Consider a query q with an unordered gold standard set Q and $k = |Q| - 1$. Let $rank_i$ be the rank of the i-th data object of Q in the ordering returned by the search operation. For evaluation purposes, we will assume that any data item not returned in the result set by the search procedure has a default rank equal to the size of the dataset. Then *average precision* is defined as follows:

$$\text{Average precision} = \frac{1}{k} \sum_{i=1}^{k} \frac{i}{rank_i}$$

Suppose $Q = \{q_1, q_2, q_3\}$ and the query is q_1. If the search results are r_1, q_2, q_3, and r_4, the average precision is $1/2 \cdot (1/2 + 2/3) = 0.583$.

6.3 Results

The evaluation is done on a PC system with a single Intel Pentium-4 3.0GHz CPU with 512KB L2 cache. The PC system has 2GB of DRAM and a 60GB 7,200RPM Maxtor disk. It runs Linux with a 2.4.20 kernel.

All results reported in this paper are average numbers obtained by running experiments multiple times.

	Methods	Average Precision	1st Tier	2nd Tier	Feature Vector Size(bits)	Sketch Size(bits)	Size ratio
VARY Image	Ferret	0.59	0.54	0.63	448	96	4.7:1
	SIMPLIcity	0.41	0.41	0.47	264	n/a	n/a
TIMIT Audio	Ferret	0.72	0.68	0.74	6,144	600	10.2:1
PSB 3D Shape	Ferret	0.32	0.30	0.41	17,472	800	21.8:1
	SHD	0.33	0.32	0.43	17,472	n/a	n/a

Table 1: Results from the search-quality benchmark suite. SIMPLIcity and SHD are domain specific tools for image and 3D shape search respectively.

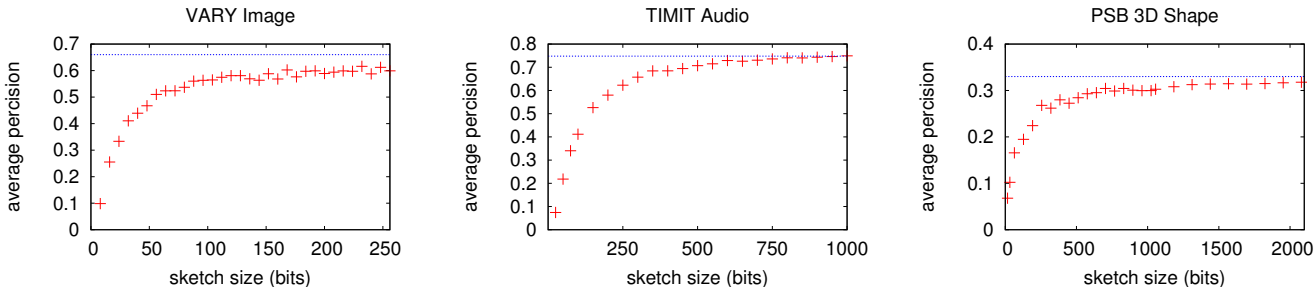

Figure 7: Average precisions of VARY image dataset, TIMIT audio dataset, and PSB 3D Shape dataset, varying the size of sketch (in bits) per feature vector. The solid line in the plots indicate the average precisions using the original feature vectors.

6.3.1 Search Quality and Speed

To answer this question, we ran experiments with the search-quality benchmark suite. For each benchmark dataset, we used the first data object in each "gold standard" similarity set as the query data object to obtain result. We also compared our results with the best known domain specific search tools. Table 1 reports our results.

The results show that all three systems achieve good search quality for the benchmarks. For the VARY image benchmark, our image search system achieves much better search quality (average precision of 0.59) than the SIMPLIcity system [45] (average precision of 0.41), which is one of the best reported image search systems in terms of metadata size and search quality. For the TIMIT audio benchmark, our system achieves average precision 0.72. We are not aware of other content-based similarity search systems for speech audio to make a direct comparison of performance on the TIMIT audio benchmark. For the PSB 3D shape models, our system achieves almost the same search quality numbers but saves storage by a factor of 22, as compared to SHD [25], which has been reported to have compact metadata and good search quality [36].

Table 2 shows the results from the search-speed benchmark suite.

Benchmark	Number of Data Objects	Avg. # Segments /Object	Avg. Search Time (s)
Mixed image	660,000	10.8	2.0
TIMIT Audio	6,300	8.6	0.09
Mixed 3D shape	40,000	1	0.01

Table 2: Results from the search-speed benchmark suite.

These results were obtained from the three similarity search systems with the sketching and filtering mechanism turned on. The search time for the Mixed 3D shape dataset is much faster since it has only one feature vector per data object.

Ideally, we would like to evaluate whether the Ferret systems can maintain the search quality and speed with much larger datasets. These are challenging questions to answer, because there is no large-scale search-quality benchmark for similarity searches. One of the main reasons is that there is no universally agreed definition of similarity. Current benchmarks are created by human subjects based on human perception of similarity. The creation of large-scale similarity search benchmarks remains an open problem.

6.3.2 Search Quality vs. Sketch Sizes

The core similarity search engine in the toolkit uses sketches as its internal data structures. The smaller the sketches, the more storage space it saves and the faster the system runs. However, smaller sketches tend to lower search quality. We are interested in determining the appropriate sketch sizes for different data types.

To answer the question, we ran the search-quality benchmark suite with different sketch sizes. For each benchmark, we measure the average precision for each sketch size. Filtering is turned off in all these experiments.

Figure 7 plots the average precisions of different sketches sizes. The data suggests that each curve has two "knee" points: low and high. The low knee point indicates the point below which (if the sketches are smaller than the low knee point), search qualities will degrade quickly. The low knee points in the data are 64-bit sketch for the VARY image benchmark, 250-bit sketch for the audio TIMIT benchmark, and 200-bit sketch for the 3D shape benchmark. The high knee point indicates the point above which (if the sketches are larger than the high knee point), search qualities will not improve much. The high knee points in the data are 88-bit sketch for the VARY image benchmark, 600-bit sketch for the audio TIMIT benchmark, and 600-bit sketch for the 3D shape benchmark.

The sketch sizes between the low and high knee points are candidates to use for each data type. In other words, the two knee points define a good sketch to original metadata size ratio range. In the VARY image benchmark case, the

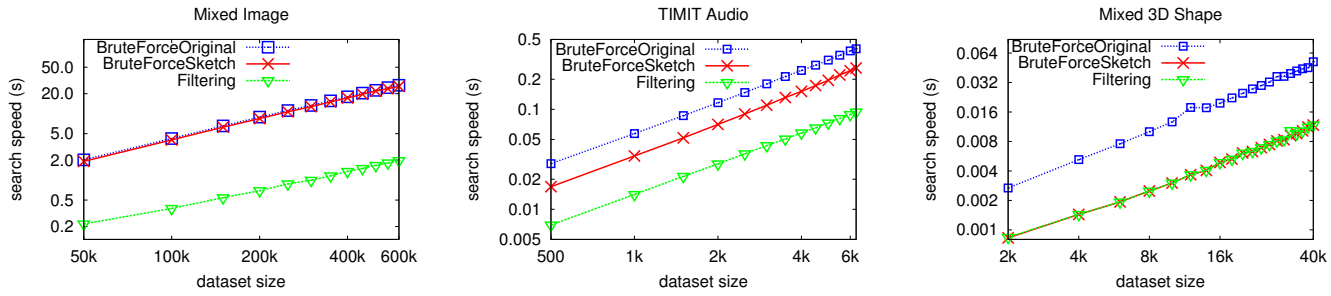

Figure 8: Query Performance for Various Search Methods

ratio range is 5:1 to 7:1. In the audio TIMIT benchmark case, the ratio range is 10:1 to 31:1. In the 3D shape benchmark, the ratio range is 29:1 to 87:1. Within the ranges, the search quality degradations are within a few percentages from those of their original metadata. This means that using sketches can reduce the metadata storage requirements by one or two orders of magnitude with small degradations of search quality.

6.3.3 Brute-Force, Sketching, Filtering

The Ferret toolkit allows users to select one of the three approaches to perform similarity searches:

- *BruteForceOriginal* This approach computes the distances between the query data object and every data object in the system, using the original feature vectors.
- *BruteForceSketch* This approach computes the distances between the query data object and every data object in the system, using the sketches.
- *Filtering* This approach uses the segment sketch to filter out very dissimilar data objects to create a candidate set and then ranks the data objects in the candidate set by computing their distances from the query data object.

We conducted our experiments with the search-speed benchmark suite. Figure 8 shows the performance trend for each benchmark by varying the size of each dataset. Our observations on the results are as follows.

The first is that the search speed of the brute-force approach is proportional to the dataset sizes, as expected. Most of the time spent on data object distance calculations. Thus, the performance is sensitive to the average number of feature vectors per data object. The average number of feature vectors per data object of the image dataset is 11, whereas that of the Mixed 3D shape models is 1. For 40,000 data objects, the search time of the brute-force approach without sketches for the Mixed image dataset is about 2 seconds, whereas that for the Mixed 3D shape dataset is only about 50 milliseconds.

The second is that using sketches not only reduces metadata size, but also improves search performance, as expected. The performance impact of using sketches is sensitive to the ratio of original data size to sketch size. The higher the ratio, the greater the impact it makes. In the Mixed image dataset case, the ratio is 5:1. There is almost no performance improvement in the results (but we do reduce the metadata size). In the Mixed shape dataset, the performance improvement is about factor of 4 since the ratio is 22:1.

The third observation is that filtering substantially improves the performance over the brute-force approach using sketches, but as expected, its search time increases proportional to the number of data objects in a dataset. Overall, filtering works well with modest dataset sizes. We believe filtering can be very useful when a user performs a similarity search in conjunction with an attribute-based search. Since the result of an attribute-based search is likely to be a smaller subset, it is natural to apply the filtering method on the attribute-based search results as a way to combine the two types of searches.

7. RELATED WORK

Although information retrieval has been extensively studied, no previous work has built a generic content-based similarity search engine for very large-scale feature-rich data. Most popular search engines such as Google, AltaVista, and Yahoo! rely on keyword based search, manual annotations, and the graph structure of hypertext documents [7, 1]. Several recent systems have focused on building a unified index for content in a personal file system which is searchable with keywords, attributes, and annotations. Apple's Spotlight [2] is an example of one such system which uses per data-type plugins to extract metadata and keywords as text suitable for indexing. MyLifeBits is another such system that manages personal information with rich annotations, both manually and automatically generated [17]. The Stuff I've Seen focuses on retrieved web pages [13]. Several desktop search tools have been released recently by Google, Yahoo!, MSN, and others. None of these systems has addressed how to perform content-based similarity search for noisy, high-dimensional, feature-rich data.

Domain-specific methods for content-based similarity search have also been studied extensively. For image similarity search, low-level features such as color, texture, and shape are usually used[14, 34, 39, 44, 35]. Recent work has shown that an effective approach is to first segment images and extract features on a per-region basis. Our recent work on image similarity search studied sketching methods as well as improved EMD distance measure for image retrieval [27], but this work did not study how to build a toolkit for large-scale and multiple kinds of feature-rich data.

Previous work on audio similarity has tended to focus on similarity search either for music [42, 6] or the spoken word. For speech audio, traditional systems use speaker independent automatic speech recognition system to generate a transcript and then apply text based search [20]. Some recent systems [30] have instead used "phone"-based approaches.

Previous work on 3D shape search has focused on the construction of shape descriptors (fixed-dimensional feature vectors) describing 3D shapes, and distance functions on shape descriptors to estimate similarity. Surveys of shape descrip-

tors and their distance functions can be found in [43, 10, 24]. Recent 3D shape search engines include [3, 4, 5].

Our similarity search engine draws on a number of recent advances in the theory community in the construction of compact data structures ("sketches") and in general dimension reduction techniques. It is often possible to produce sketches such that a particular distance function on the original data may be quickly estimated from the corresponding sketches with provable bounds on the error. Early work on sketches include the min-wise independent permutation sketches for filtering near-duplicate documents that Broder *et al.* developed for the AltaVista search engine [9, 8]. Subsequent work by Indyk and Motwani [22] introduced the notion of locality-sensitive hash functions selected so that the collision probability is higher for pairs of objects that are closer in some suitable sense. Such families are useful for constructing compact data structures for nearest-neighbor search. Kushilevitz, Ostrovsky and Rabani [26] developed a hashing scheme to distinguish between pairs of objects with ℓ_1 distances above and below a given threshold – we adapt their ideas in our sketch construction. Charikar [11] as well as Indyk and Thaper [23] developed compact representations suitable for approximating the Earth Mover's Distance. Such schemes were recently used by Grauman and Darrell [18] for efficient matching of contours and images. For an overview of dimension reduction techniques, please see Imola Fodor's survey [15]. A natural question that comes to mind is how our sketching scheme compares with locality sensitive hashing [22]. Such a comparison is not well defined since locality sensitive hash functions are designed for an indexing approach, instead of the filtering approach we take. Our use of sketches is to construct a compact representation for accurate estimation of feature distances rather than for indexing. Further, constructions of locality sensitive hash functions depend on the underlying metric of interest and have been devised for normed spaces such as ℓ_1 and ℓ_2. We use a more complicated distance function on feature vectors and use Earth Mover's Distance to compute the distance between sets of feature vectors, so these results are not directly applicable to our setting.

8. CONCLUSION AND FUTURE WORK

In this paper, we present the design and implementation of the Ferret content-based similarity search toolkit. Using this toolkit, we have built similarity search systems for image, audio, and 3D shape model data. An external research group has also used the toolkit for genomic data analysis. Our experience has shown that it is straightforward to use the toolkit for a variety of feature-rich data. Furthermore, our experimental results show that the systems built with the Ferret toolkit can achieve high-quality similarity search at reasonably high-speed on common domain-specific benchmark datasets.

We have shown that using sketches can reduce metadata storage requirements significantly with minimal degradation of similarity search quality. For the VARY image benchmark, using a 96-bit sketch per feature vector (or about 94 bytes per image) reduces average precision by about 11%, but uses only 1/5 of the original metadata size. For the TIMIT audio benchmark, using a 600-bit sketch per word reduces average precision by about 4%, but uses about 1/10 of the original metadata size. For the 3D shape model benchmark, using an 800-bit sketch per descriptor degrades the average precision by about 3%, but uses only 1/22 of the original metadata size.

Figure 9: A screen shot of the web interface of the 3D shape similarity search system constructed by using the Ferret toolkit. It shows the query results of the first model (shown in yellow). Each result includes name of the model file, and its distance to the query model.

The current implementation of the Ferret toolkit represents a first step toward building high-performance content-based similarity search engine. Outstanding research problems include the design and implementation of improved indexing data structures for similarity search, more effective and efficient distance functions, and the design of sketching algorithms for these functions. For instance, the improved EMD distance function is an effective distance measure for image and audio data, but it is relatively inefficient to compute. We plan to explore more efficiently computable distance functions and to develop appropriate sketch algorithms for them.

The current version of the toolkit uses filtering on sketches to produce a small candidate set for subsequent ranking. Although the use of sketches provides a substantial performance benefit, we expect to investigate more efficient out-of-core indexing data structures for similarity search to further improve support for very large data sets. We also expect to continue expanding the usage of Ferret toolkit to include video and other sensor data and to release the toolkit to the public domain.

Acknowledgments

This project is sponsored in part by a Google research grant, by National Science Foundation grants EIA-0101247, CNS-0509447, CCR-0205594, CCR-0237113 and and by a Microsoft Research grant. William Josephson is supported by a National Science Foundation Graduate Research Fellowship.

We would like to thank Perry Cook and Matt Hoffman for their help with audio data segmentation and feature extraction, and thank Matt Hibbs and Olga Troyanskaya for being the users of the toolkit.

9. REFERENCES

[1] Altavista. `http://www.altavista.com`.
[2] Spotlight: Find anything on your mac instantly. `http://images.apple.com/macosx/pdf/MacOSX_Spotlight_TB.pdf`.

[3] 3D model retrieval. http://amp.ece.cmu.edu/projects/3DModelRetrieval/.

[4] 3D model retrieval. http://3d.csie.ntu.edu.tw/~dynamic/.

[5] 3D model retrieval. http://shape.cs.princeton.edu/search.html.

[6] A. Berenzweig and D. Ellis. Locating singing voice segments within music signals. In *Proc. of IEEE Workshop on Applications of Signal Processing to Acoustics and Audio*, October 2001.

[7] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc. of the 7th World Wide Web Conference*, 1998.

[8] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer Systems and Sciences*, 60(3):630–659, 2000.

[9] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proc. of the Sixth Int. World Wide Web Conf.*, pages 391–404, 1997.

[10] 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.

[11] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. of the 34th Annual ACM Symp. on Theory of Computing*, pages 380–388, 2002.

[12] Y. Deng and B. S. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2001.

[13] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff I've seen: A system for personal information retrieval and re-use. In *Proc. of the 26th ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 72–79, 2003.

[14] J. P. Eakins and M. e. Graham. Content-based image retrieval: A report to the JISC technology applications programme. Technical report, University of Northumbria at newcastle, Institute for Image Data Research, 1999.

[15] I. K. Fodor. A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory, 2002.

[16] 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.

[17] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. Mylifebits: Fulfilling the Memex vision. In *Proc. of ACM Multimedia Conference*, pages 235–238, 2002.

[18] K. Grauman and T. Darrell. Fast contour matching using approximate earth mover's distance. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, 2004.

[19] J. Gray and A. S. Szalay. Where the rubber meets the sky: Bridging the gap between databases and science. *IEEE Data Engineering Bulletin*, 27(4):3–11, December 2004.

[20] A. Hauptmann, R. Jones, K. Seymore, S. Slattery, M. Witbrock, and M. Siegler. Experiments in information retrieval from spoken documents. In *In Proc. of the Broadcast News Transcription and Understanding Workshop*, pages 175–181, 1998.

[21] ftp://db.stanford.edu/pub/wangz/image.vary.jpg.tar.

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

[23] P. Indyk and N. Thaper. Fast image retrieval via embeddings. In *Proc. of the 3rd Int. Workshop on Statistical and Computational Theories of Vision*, 2003.

[24] 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.

[25] 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.

[26] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal of Computing*, 30(2):457–474, 2000.

[27] 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.

[28] P. Lyman, H. Varian, K. Swaringen, P. Charles, N. Good, L. Jordan, and J. Pal. How much information 2003? http://www.sims.berkeley.edu/research/projects/how-much-info-2003.

[29] W. Ma and H. Zhang. Benchmarking of image features for content-based retrieval. In *Proc. of IEEE 32nd Asilomar Conf. on Signals, Systems, Computers*, volume 1, pages 253–257, 1998.

[30] N. Moreau, H. G. Kim, and T. Sikora. Phone-based spoken document retrieval in conformance with the mpeg-7 standard. *Proc. of the Audio Engineering Society 25th Intl. Conf.*, 2004.

[31] M. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proc. of the 1999 Summer USENIX Technical Conf.*, June 1999.

[32] L. Rabiner and M. Sambur. An algorithm for determining the endpoints of isolated utterances. *Bell System Technical Journal*, 54:297–315, 1975.

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

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

[35] R. Schettini, G. Ciocca, and S. Zuffi. A survey of methods for color image indexing and retrieval in image databases. *Color Imaging Science: Exploiting Digital Media*, 2001.

[36] P. Shilane, M. Kazhdan, P. Min, and T. Funkhouser. The Princeton shape benchmark. In *Proc. of the Conf. on Shape Modeling and Applications*, 2004.

[37] M. Siegler and M. Witbrock. Improving the suitability of imperfect transcriptions for information retrieval of spoken documents. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 1999.

[38] *Sets of Similar Images*. http://dbvis.inf.uni-konstanz.de/research/projects/SimSearch/effpics.html.

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

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

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

[42] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5), July 2002.

[43] R. C. Veltkamp. Shape matching: Similarity measures and algorithms. In *Proc. of the Int. Conf. on Shape Modeling & Applications*, page 188, 2001.

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

[45] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics-sensitive integrated matching for picture libraries. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(9):947–963, 2001.

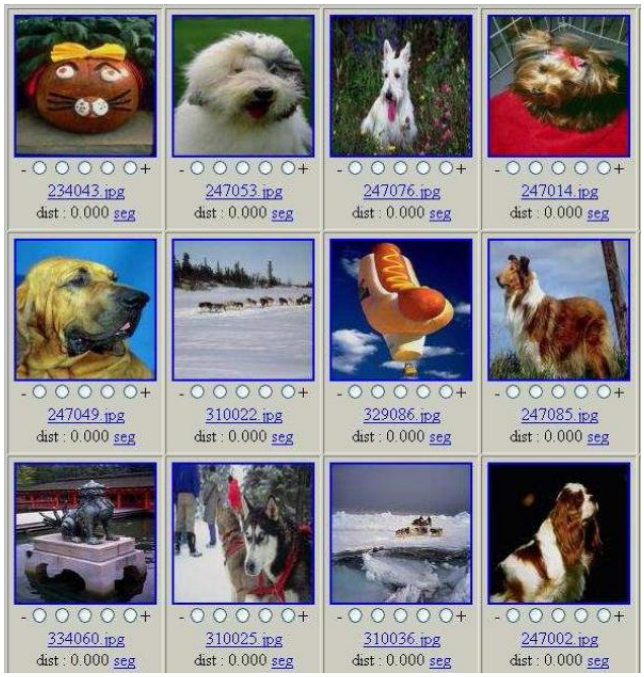

Figure 10: The image similarity search web interface showing the result of an attribute-based search for the keyword "dog" in the "Corel" image collection.

Figure 11: The result of image similarity search using the second dog photo in the top row of the attribute-based search results in the figure above.

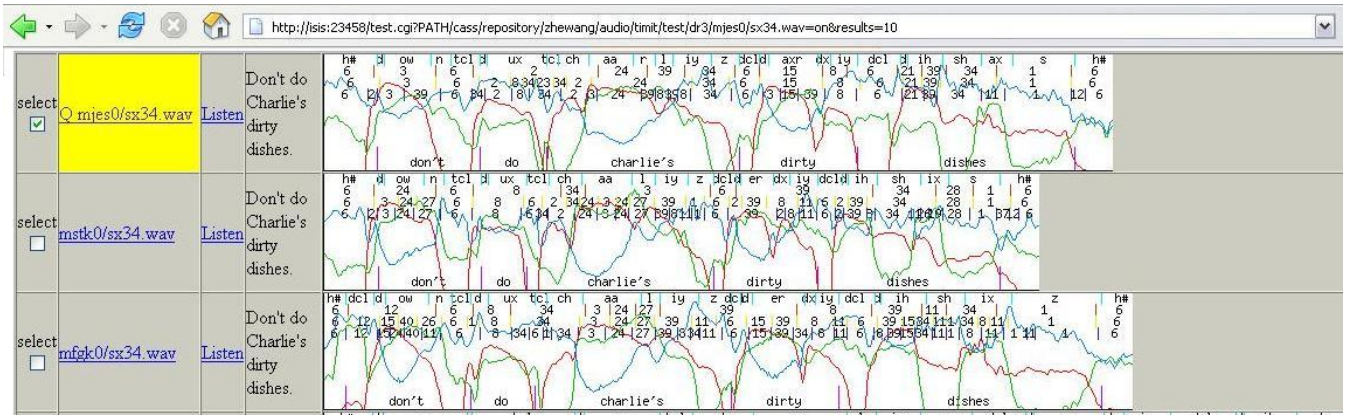

Figure 12: The audio similarity search web interface showing the results of a sample search. Each row shows a similar audio file, a link to listen to the audio file, transcribed text, as well as the curves of its first three MFCC parameters.

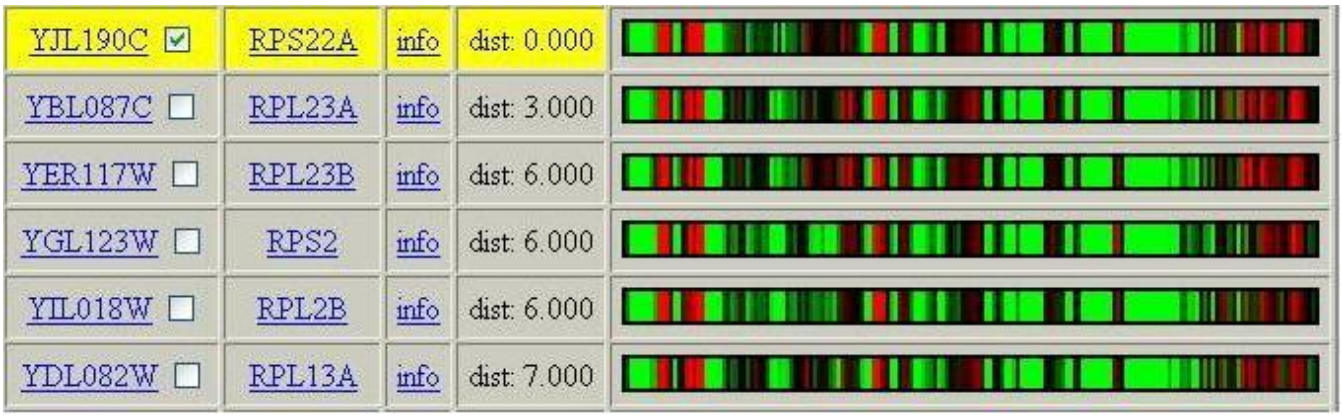

Figure 13: The genomic similarity search of the web interface showing the results of a search for the gene YJL190C. Each row shows one similar gene, its gene/ORF names, a link to the gene's information, its distance to the query gene, and its colored expression visualization.