Filtering Methods for Similarity-Based Multimedia Retrieval

Vassilis Athitsos, Jonathan Alon, Stan Sclaroff, and George Kollios

Computer Science Department, Boston University, Boston MA 02215, USA
{athitsos, jalon, sclaroff, gkollios}@cs.bu.edu

Abstract. A common problem in multimedia databases is retrieving the most similar matches to a query object. Finding those matches can be too slow to be practical, especially in domains where comparing multimedia objects involves computationally expensive similarity (or distance) measures. Filter-and-refine retrieval is a framework for addressing this problem: the filter step quickly filters out most database objects, and the refine step identifies the best matches among the remaining candidates. This paper describes two filtering methods, that work by constructing efficient approximations of computationally expensive similarity measures. The first method can be applied to arbitrary domains, and the second method explicitly targets domains where measuring similarity includes an alignment process. The benefits of these two filtering methods are illustrated in experiments with databases from different domains, i.e., hand images, gesture videos, and online digit recognition for hand-held devices.

1 Introduction

A common problem in multimedia databases is retrieving the most similar matches to a query object. For example, in a database of video sequences, we may want to use a video sequence as a query, and have the system retrieve sequences that are similar to the query. Other domains where this type of similarity query is useful include databases of photographs, songs, or medical images. In order to answer such queries, we first need to define a domain-specific quantitative measure of similarity between two objects. Given such a measure, the most naive way of retrieving the most similar database matches is to do brute-force search, i.e. to compare the query object to every single database object.

The problem with brute-force search is that it can be too slow for practical applications. This problem is exacerbated by the fact that many useful similarity measures are computationally expensive. Examples of such expensive measures are Dynamic Time Warping for time series (for example for video or audio data) [12], and the chamfer distance [3] or the shape context distance [4] for edge images. Filter-and-refine retrieval is a framework for speeding up similarity-based retrieval under computationally expensive similarity measures. In this framework, retrieval is performed in two steps. First we do a filtering step, in which we employ computationally efficient pruning methods to select, among all database objects, a small set of likely matches to the query. Then we refine the results of the filtering step, by applying the computationally expensive measure only between the database and the likely matches identified at the filter step. Naturally, in order to apply this framework in a particular domain, we need to define appropriate filtering methods for that domain, i.e., computationally efficient methods that can quickly and reliably identify a small set of candidate matches, while rejecting the rest of the database.

This paper describes two general filtering methods for filter-and-refine retrieval. The first method is called BoostMap, and its formulation is applicable in arbitrary domains. The BoostMap algorithms defines an embedding, which maps multimedia objects into a real vector space. Instead of measuring distances in the original space, which is too slow, using the embedding we can measure distances in the vector space, which is significantly more efficient.

The second filtering method that we describe is applicable in domains where an expensive alignment between the query object and database objects is required in order to measure their similarity. Our method produces an approximate alignment between two objects, based on their exact alignment to a small number of prototypes. Using this approximate alignment we can quickly identify, for each query object, a small set of potential database matches.

The benefits of these two filtering methods are illustrated in experiments with databases from different domains, i.e., hand images, gesture videos, and online digit recognition for hand-held devices.

2 Related Work and Background

Various methods have been employed for similarity indexing in image and video databases, including hashing and tree structures [17]. However, these methods typically rely on Euclidean or metric properties, and are not appropriate for arbitrary non-metric spaces. Approximate nearest neighbor methods [11] proposed in the literature are also only applicable to some specific metrics.

In domains where the distance measure is computationally expensive, significant computational savings can be obtained by constructing a distance- approximating embedding, which maps objects into another space with a more efficient distance measure. A number of methods have been proposed for embedding arbitrary metric spaces into a Euclidean or pseudo-Euclidean space [5, 7, 10, 13, 15, 16, 18]. Among the above methods, Lipschitz embeddings [9], FastMap [7], MetricMap [16] and SparseMap [10], can be used to speed up similarity-based retrieval. Those four methods are the most related to the methods that we propose in this paper. Our goal in designing the new methods has been to achieve better retrieval performance, i.e., faster and more accurate, than those four methods can achieve.

2.1 Background

Let X be a set of objects, and $D_X(x_1, x_2)$ be a distance measure between objects $x_1, x_2 \in X$. D_X can be metric or non-metric. A Euclidean embedding $F : X \rightarrow$

$\mathbb{R}^d$ is a function that maps objects from X into the d-dimensional Euclidean space $\mathbb{R}^d$, where distances are typically measured using an L_p or weighted L_p measure, denoted as $D_{\mathbb{R}^d}$.

Given an object $r \in X$, a simple 1D Euclidean embedding F^r can be defined as follows:

$$F^r(x) = D_X(x, r) \ . \tag{1}$$

The object r that is used to define F^r is typically called a *reference object* or a *vantage object* [9].

Alternative formulas for defining 1D embeddings can be found in the literature [7, 9]. Lipschitz embeddings [9] and FastMap [7] are methods for constructing multidimensional embeddings, using 1D embeddings as building blocks. For example, if $r_1, \ldots, r_d$ are d reference objects, we can define a d-dimensional Lipschitz embedding embedding $F : X \to \mathbb{R}^d$ as: $F(x) = (F^{r_1}(x), \ldots, F^{r_d}(x))$.

Given such an embedding F, we can use it for the filter step of filter-and-refine retrieval [9]. Given a query object q, we compute $F(q)$, which involves measuring the distance D_X between q and d reference objects. Then, at the filter step, we identify, for some integer $p > 0$, the p database objects that F maps the closest to $F(q)$. At the refine step we identify the best matches for the query, by measuring the exact distance D_X between the query and the p objects selected at the filter step.

If the database has n objects, brute-force retrieval identifies the nearest neighbors of q by measuring the distance D_X between q and each database object, i.e., by performing n evaluations of D_X. Filter-and-refine retrieval, in the above example, performs only $d+p$ evaluations of D_X, and performs (at the filter step) n comparisons of vectors in Euclidean space. In domains where D_X is computationally expensive, filter-and-refine retrieval can be orders of magnitude faster than brute-force search, if $d + p \ll n$.

3 Learning Embeddings with the BoostMap Algorithm

Suppose we have an embedding F with the following property: for any $q, a, b \in X$, if q is closer to a than to b, then $F(q)$ is closer to $F(a)$ than to $F(b)$. We can easily derive that F would also have the following property: for every query q, and every integer $k > 0$, if a is the k-nearest neighbor of q in the database, then $F(a)$ is the k-nearest neighbor of $F(q)$ among the embeddings of all database objects. Such an embedding would lead to perfectly accurate retrieval, without even needing an extra refinement step.

Finding such a perfect embedding is usually impossible. However, we can try to construct an embedding that, as much as possible, tries to behave like a perfect embedding. In other words, we want to construct an embedding in a way that maximizes the fraction of triples (q, a, b) for which the embedding indeed preserves the relative ranking of a and b with respect to q.

More formally, using an embedding F we can define a classifier $\tilde{F}$, that estimates (sometimes wrongly) for any three objects q, a, b if q is closer to a or to b.

$\tilde{F}$ is defined as follows:

$$\tilde{F}(q, a, b) = \|F(q) - F(b)\| - \|(F(q) - F(a)\| \ . \quad (2)$$

A positive value of $\tilde{F}(q, a, b)$ means that F maps q closer to a than to b, and can be interpreted as a "prediction" that q is closer to a than to b in the original space X. If this prediction is always correct, then F perfectly preserves the similarity structure of X.

Simple 1D embeddings, like the one defined in Eq. 1, are expected to behave as *weak classifiers*, i.e. classifiers that may have a high error rate, but at least give answers that are not as bad as random guesses. Given many weak classifiers, a well-studied problem in machine learning is how to combine such classifiers into a single, strong classifier, i.e., a classifier with a low error rate. A popular choice is AdaBoost [14], which has been successfully applied to several domains in recent years.

In [1] we introduced the BoostMap algorithm, which uses AdaBoost to construct an embedding. The input to AdaBoost is a large set of randomly picked 1D embeddings, and a large set of training triples (q, a, b) of objects, for which we know if q is closer to a or to b. The output of AdaBoost is a classifier $H = \sum_{j=1}^{d} \alpha_j \tilde{F}_j$, where each $\tilde{F}_j$ is the weak classifier associated with a 1D embedding F_j. If AdaBoost has been successful, then H has a low error rate.

Using H, we can easily define a high-dimensional embedding F_{out} and a distance measure $D_{\mathbb{R}^d}$ with the following property: for any triple (q, a, b), if q is closer to a than to b, H misclassifies that triple if and only if, according to distance measure $D_{\mathbb{R}^d}$, $F_{\text{out}}(q)$ is closer to $F_{\text{out}}(b)$ than to $F_{\text{out}}(a)$. We define F_{out} and $D_{\mathbb{R}^d}$ as follows:

$$F_{\text{out}}(x) = (F_1(x), ..., F_d(x)) \ . \quad (3)$$

$$D_{\mathbb{R}^d}(F_{\text{out}}(x), F_{\text{out}}(y)) = \sum_{j=1}^{d} (\alpha_j |F_j(x) - F_j(y)|) \ . \quad (4)$$

It is easy to prove that H and F_{out} fail on the same triples. Therefore, if AdaBoost has successfully produced a classifier H with low error rate, then F_{out} inherits the low error rate of H, and is expected to give highly accurate results at the filtering step.

The exact training algorithm is described in [1].

4 Defining Embeddings for Unaligned Objects

In some multimedia databases, objects are represented by sets of observations of different size, or by sequences of observations of varying length. This is typical in computer graphics databases, where the length corresponds to the model size, e.g., the number of 2D or 3D points, and in speech and video databases, or more generally in multidimensional time-series databases, where the sequence length corresponds to the period of time in which the object persists. In such databases, it is often necessary to find the correspondence between the observations of the query object and the database objects prior to computing a (dis)similarity score.

The number of possible alignments is usually exponential to the number of features in each object. Efficient algorithms, like Dynamic Time Warping [12] or bipartite matching, can find an optimal alignment in polynomial time, but these algorithms can still be computationally expensive in practice. General embedding methods like BoostMap, Lipschitz embeddings [9] or FastMap [7] can be applied in such domains, in order to define an efficient approximation of the similarity measure. However, by taking advantage of the structure of such domains we will define a more specific embedding method, that tends to use a relatively small number of reference objects.

4.1 Approximate Alignment Using Correspondences to Reference Objects

Let X be a space of such objects, and $Q, R \in X$ be two objects of size n and m respectively:

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n \ , \tag{5}$$

$$R = r_1, r_2, \ldots, r_j, \ldots, r_m \ . \tag{6}$$

Each q_i and r_j is an observation belonging to a feature space G, with distance measure D_G.

We assume that we have an alignment algorithm (like Dynamic Time Warping [12], or bipartite matching) that establishes correspondences between elements of Q and elements of R. Given Q and R, this algorithm produces an alignment $W(Q, R)$, where the k^{th} element of $W(Q, R)$, denoted as $w_k = (q_i, r_j)$, indicates that observation q_i of Q corresponds to observation r_j of R. We assume that each r_j occurs in at least one w_k in $W(Q, R)$. In many domains, the distance $D_X(Q, R)$ is defined as follows:

$$D_X(Q, R) = \sum_{k=1}^{K} D_G(w_k) \ . \tag{7}$$

That is, the distance between Q and R is the sum of distances between each pair q_i and r_j of corresponding observations that appears in alignment $W(Q, R)$.

Let R be an arbitrary object in X. We can use R as a *reference object* [9], to define an embedding $F^R : X \to G^m$, where m is the size of object R. In simple terms, F maps any object Q into an ordered set of m features, so that $F^R(Q) = (q'_1, q'_2, \ldots, q'_m)$. Each q'_j is simply the "average" (i.e., arithmetic mean where applicable, although in some domains we need to define some alternative definition of average) of all features q_i that map to r_j under alignment $W(Q, R)$.

Naturally, if we choose d reference objects R_i, we can define an embedding F by simply concatenating embeddings F^{R_i}, i.e. $F(Q) = (F^{R_1}(Q), \ldots, F^{R_d}(Q))$. F maps objects into $G^{m'}$, where m' is the sum of the sizes of the R_i's. Now, if

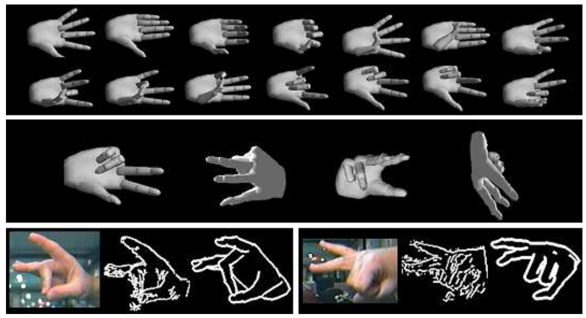

Fig. 1. Top: 14 of the 26 hand shapes used to generate the hand database. Middle: four of the 4128 3D orientations of a hand shape. Bottom: for two test images we see, from left to right: the original hand image, the extracted edge image that was used as a query, and a correct match (noise-free computer-generated edge image) retrieved from the database.

$F(Q) = (q'_1, \ldots, q'_{m'})$ and $F(S) = (s'_1, \ldots, s'_{m'})$, we can define the distance D' between them as:

$$D'(F(Q), F(S)) = \sum_{j=1}^{m'} D_G(q'_j, s'_j) \ . \tag{8}$$

Intuitively, the computationally expensive way to compare Q and S is to first find their alignment $W(Q, S)$ (which is assumed to be time consuming) and then evaluate $D_X(Q, S)$ based on this alignment. The embedding F maps objects into ordered sets of features, which are naturally aligned to each other: each q'_j of $F(Q)$ maps to s'_j of $F(S)$. This alignment is only an approximation of the alignment $W(Q, S)$, and it is based on the fact that both q'_j and s'_j were aligned to the same feature of some reference object R_i. Since $F(Q)$ and $F(S)$ are automatically aligned to each other, comparing them using distance measure D' is straightforward, and takes time linear in m'. Therefore, D' is a computationally efficient distance measure that we can use for the filter step in filter-and-refine retrieval [9].

Reference objects can be chosen using sequential forward search, which is essentially a greedy strategy: we pick the first reference object to be the one that gives the best results when used by itself, and we pick the i-th reference object to be the one that gives the best results when combined with $R_1, \ldots, R_{i-1}$.

5 Experiments

We have experimentally evaluated the two techniques described in this paper, i.e., BoostMap and the method for approximate alignment.

We used two datasets to evaluate BoostMap: a database of hand images, and an ASL (American Sign Language) database, containing video sequences of

ASL signs. The hand database contains 107,328 hand images, generated using computer graphics. 26 hand shapes were used to generate those images. Each shape was rendered under 4128 different 3D orientations (Figure 1). As queries we used 703 real images of hands. Similarity between hand images is evaluated using the symmetric chamfer distance [3], applied to edge images. Evaluating the exact chamfer distance between a query and the entire database takes about 260 seconds.

The ASL database contains 880 gray-scale video sequences. Each video sequence depicts a sign, as signed by one of three native ASL signers. As queries we used 180 video sequences of ASL signs, signed by a single signer who was not included in the database. Similarity between video sequences is measured as follows: first, we use the similarity measure proposed in [6], which is based on optical flow, as a measure of similarity between single frames. Then, we use Dynamic Time Warping [12] to compute the optimal time alignment and the overall matching cost between the two sequences. Evaluating the exact distance between the query and the entire database takes about six minutes.

In the experiments, we compare BoostMap to FastMap [7]. For both BoostMap and FastMap we evaluated embeddings of many different dimensions, and for each accuracy level (95% or 100%) we experimentally found the optimal settings (with respect to retrieval time) of two parameters: embedding dimensionality and number of objects to be kept at the filter step. All the results we report were obtained using those optimal settings. Note that, given an embedding and a number of dimensions, we can achieve any accuracy level we desire by simply selecting a large enough number of objects at the filter step. For example, if the filter step does not filter out any objects, we always get 100% accuracy. The reason that we want embeddings of higher quality is that those embeddings will allow us to achieve the desired accuracy while selecting a smaller number of objects at the filter step, thus reducing the processing time for the refine step.

Table 1 shows the results of BoostMap vs. FastMap. For the hand database, BoostMap leads to significantly faster retrieval. In the ASL database BoostMap gives slightly better results than FastMap.

In order to evaluate the approximate alignment method described in Section 4, we used the isolated digits benchmark (category 1a) of the UNIPEN Train-R01/V07 online handwriting database [8], which consists of $15,953$ digit samples. Each digit is represented as a a series of so-called pen-down and pen-up components. Each component contains a sequence of pen tip information which is sampled from the writer's pen movement.

Data preprocessing and feature extraction is carried out exactly as described in [2]. Data preprocessing consists of removing pen-up components and concatenating the remaining pen-down components into a single sequence of (x,y) locations, and removing repetitions of consecutive observations. Feature extraction consists of normalizing the (x,y) coordinates by subtracting the mean and dividing by standard deviation, and then adding a third directional feature, which is the tangent angle of the vector connecting two consecutive locations.

Table 1. Comparison of BoostMap, FastMap and using brute-force search, for the purpose of retrieving the exact nearest neighbors successfully for 95% or 100% of the queries, using filter-and-refine retrieval D_X # per query is the total number of D_X computations needed per query, in order to embed the query and rank the top p candidates selected at the filter step. The exact D_X column shows the results for brute-force search, in which we simply evaluate D_X distances between the query and all database images.

Retrieval accuracy and efficiency for hand database					
Method	BoostMap		FastMap		Exact D_X
accuracy	95%	100%	95%	100%	100%
D_X # per query	823	4267	3864	17518	107328
seconds per query	2.3	10.6	9.4	42.4	260

Retrieval accuracy and efficiency for ASL database					
Method	BoostMap		FastMap		Exact D_X
accuracy	95%	100%	95%	100%	100%
D_X # per query	249	375	269	398	880
seconds per query	103	155	111	164	363

In summary, a feature vector sequence is defined as $Q = (q_1, q_2, \ldots, q_i, \ldots, q_n)$, where each $q_i = (\tilde{x}_i, \tilde{y}_i, \theta_i)$.

For the experiment the digits were randomly and disjointly divided into training and test sets of ratio about 2:1 (or 10,630 : 5,323 samples). The distance measure D_X used for classification is Dynamic Time Warping [12]. In the filter step, we precomputed the alignment between all database (training) digits and 10 reference digits. Thus, each database digit was embedded into a $1,101$ dimensional space (the total sequence length of the 10 reference digits is 367, and each sample consists of 3 features $(\tilde{x}, \tilde{y}, \theta)$). During the online phase, a test digit was embedded into the $1,101$ dimensional space in a similar way. The L_1 distance in this vector space was used for approximating the true DTW distance at the filter step. The filter step selected 100 candidate matches, and the refine step computed the exact DTW measure between the test object and the 100 candidates. In Table 2 we show the error rate and classification time for brute-force search, the filtering step with no refinement, and filter-and-refine retrieval. Filter-and-refine retrieval leads to minimal loss in accuracy, while achieving a speed-up factor of about 28 over brute-force search.

6 Discussion

The two methods described in this paper can be applied in a wide range of domains where similarity or distance between objects is computationally expensive to evaluate. We demonstrate the usefulness of these methods with experiments on three databases from different domains. In all experiments we achieve signifi-

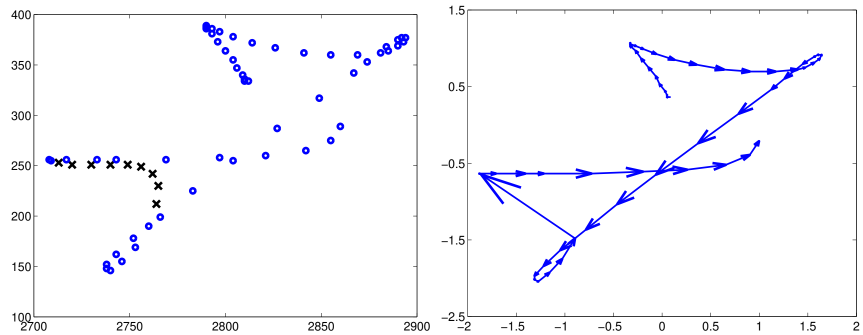

Fig. 2. Left: Example of a "seven". Circles denote "pen-down" locations, x's denote "pen-up" locations. Right: The same example, after preprocessing.

Table 2. Comparison of Filter, Filter&Refine and Exact DTW, for the purpose of nearest neighbor classification. DTW # per query is the total number of DTW computations needed per query, in order to embed the query using $R = 10$ reference digits and rank the top $p = 100$ candidates selected at the filter step. The Exact DTW column shows the results for brute-force search, in which we simply evaluate DTW distances between the query and all database digits .

NN classification error and efficiency for UNIPEN digits database			
Method	Filter	Filter&Refine	Exact
NN-error	2.33%	2.09%	2.05%
DTW # per query	10	110	10,630
seconds per query	0.32	0.42	11.95

cant speed-ups over brute force search, and improved performance compared to alternative methods.

At the same time, it is important to identify a larger number of datasets, so that we can thoroughly evaluate our algorithms and other existing approaches and identify their relative strengths and weaknesses. Examples of computationally expensive measures on which our methods can be applied include the Hausdorff distance, the Earth Mover's distance, the edit distance for matching strings and sequences (like proteins and DNA), or the Kullback-Leibler distance for matching probability distributions. Efficient filtering methods, like the ones proposed in this paper, can help real systems benefit from the precision of such expensive distance measures, while keeping retrieval complexity manageable.

Acknowledgments

This research was funded in part by the U.S. National Science Foundation, under grants IIS-0208876, IIS-0308213, IIS-0329009, and CNS-0202067, and the U.S. Office of Naval Research, under grant N00014-03-1-0108.

References

1. V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: A method for efficient approximate similarity rankings. In *CVPR*, 2004.
2. C. Bahlmann and H. Burkhardt. The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping. *PAMI*, 26(3), 2004.
3. H.G. Barrow, J.M. Tenenbaum, R.C. Bolles, and H.C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *IJCAI*, pages 659–663, 1977.
4. S. Belongie, J. Malik, and J. Puzicha. Matching shapes. In *ICCV*, volume 1, pages 454–461, 2001.
5. J. Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.
6. A.A. Efros, A.C. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *ICCV*, pages 726–733, 2003.
7. C. Faloutsos and K.I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD*, pages 163–174, 1995.
8. I. Guyon, L. Schomaker, and R. Plamondon. Unipen project of on-line data exchange and recognizer benchmarks. In *12th International Conference on Pattern Recognition*, pages 29–33, 1994.
9. G.R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *PAMI*, 25(5):530–549, 2003.
10. G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins. Technical Report 99-50, CS Department, Rutgers University, 1999.
11. P. Indyk. *High-dimensional Computational Geometry*. PhD thesis, Stanford University, 2000.
12. Eamonn Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.
13. S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.
14. R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
15. J.B. Tenenbaum, V. de Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.
16. X. Wang, J.T.L. Wang, K.I. Lin, D. Shasha, B.A. Shapiro, and K. Zhang. An index structure for data mining and clustering. *Knowledge and Information Systems*, 2(2):161–184, 2000.
17. D.A. White and R. Jain. Similarity indexing: Algorithms and performance. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 62–73, 1996.
18. F.W. Young and R.M. Hamer. *Multidimensional Scaling: History, Theory and Applications*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.