Query-Sensitive Embeddings

VASSILIS ATHITSOS
Boston University
MARIOS HADJIELEFTHERIOU
AT&T Labs-Research
GEORGE KOLLIOS
Boston University
and
STAN SCLAROFF
Boston University

A common problem in many types of databases is retrieving the most similar matches to a query object. Finding these matches in a large database can be too slow to be practical, especially in domains where objects are compared using computationally expensive similarity (or distance) measures. Embedding methods can significantly speed-up retrieval by mapping objects into a vector space, where distances can be measured rapidly using a Minkowski metric. In this article we present a novel way to improve embedding quality. In particular, we propose to construct embeddings that use a *query-sensitive* distance measure for the target space of the embedding. This distance measure is used to compare those vectors that the query and database objects are mapped to. The term "query-sensitive" means that the distance measure changes, depending on the current query object. We demonstrate theoretically that using a query-sensitive distance measure increases the modeling power of embeddings and allows them to capture more of the structure of the original space. We also demonstrate experimentally that query-sensitive embeddings can significantly improve retrieval performance. In experiments with an image database of handwritten digits and a time-series database, the proposed method outperforms existing state-of-the-art non-Euclidean indexing methods, meaning that it provides significantly better tradeoffs between efficiency and retrieval accuracy.

Categories and Subject Descriptors: H.3.1 [**Informaton Strorage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.2.4 [**Database Management**]: Systems—*Multimedia databases*

General Terms: Algorithms, Theory

This work was supported by NSF Grants IIS-0308213 and IIS-0133825, and by ONR Grant N00014-03-1-0108.

Author's addresses: V. Athitsos (contact author), G. Kollios, and S. Sclaroff, Computer Science Department, Boston University, 111 Cummington Street, Boston, MA 02215; email: athitsos@cs.bu.edu; M. Hadjieleftheriou, AT&T Labs-Research, 180 Park Avenue, Florham Park, NJ 07932.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 0362-5915/2007/06-ART8 $5.00 DOI 10.1145/1242524.1242525 http://doi.acm.org/10.1145/1242524.1242525

Additional Key Words and Phrases: Embedding methods, similarity matching, nearest-neighbor retrieval, non-Euclidean spaces, nonmetric spaces

ACM Reference Format:

Athitsos, V., Hadjieleftheriou, M., Kollios, G., and Sclaroff, S. 2007. Query-sensitive embeddings. ACM Trans. Datab. Syst. 32, 2, Article 8 (June 2007), 37 pages. DOI = 10.1145/1242524.1242525 http://doi.acm.org/ 10.1145/ 1242524.1242525

1. INTRODUCTION

A common problem in many types of databases is retrieving the most similar matches to a query object. Finding these matches in a large database can be too slow to be practical, especially in domains where objects are compared using computationally expensive similarity (or distance) measures. Although numerous indexing methods have been proposed for speeding-up nearest-neighbor retrieval [Böhm et al. 2001; White and Jain 1996], the majority of such methods typically assume that we are operating in a Euclidean space, or so-called "coordinate space," where each object is represented as a feature vector of fixed dimensions. In many actual applications, these assumptions are not obeyed because we need to use distance measures that are non-Euclidean and even nonmetric, and because the objects are not of fixed dimensionality. Examples of computationally expensive non-Euclidean distance measures include the Kullback-Leibler distance for matching probability distributions, dynamic time warping for matching time-series, bipartite matching for comparing two sets of features, or the edit distance for matching strings and biological sequences. It is important to design efficient methods for nearest-neighbor retrieval in such spaces.

A number of methods for efficient retrieval in non-Euclidean spaces utilize embeddings, that is, functions that map objects into a real vector space. These mappings aim to preserve a large amount of the proximity structure of the original space, so that nearby objects tend to get mapped to nearby vectors. Embeddings can significantly speed-up nearest-neighbor retrieval when measuring distances between vectors is faster than comparing objects in the original space. Comparing vectors takes time linear to the dimensionality of the vectors, whereas non-Euclidean distance measures often have time complexity that is superlinear to the size of the representation of the objects. For example, dynamic time warping has quadratic complexity and bipartite matching has cubic complexity.

For an embedding to be useful for nearest-neighbor retrieval, the embedding should tend to map near neighbors in the original space to near neighbors in the target space. The more an embedding adheres to this requirement, the more useful the embedding becomes for retrieval purposes. Retrieval performance is evaluated in terms of accuracy and efficiency: We want to retrieve the true nearest- neighbors of the query as often as possible, and we want retrieval to be as fast as possible.

In this article we present a novel way to improve embedding quality. In particular, we propose to construct embeddings that use a *query-sensitive* distance measure for the target space of the embedding. This distance measure is used

to compare those vectors that the query and database objects are mapped to. The term "query-sensitive" means that the distance measure changes, depending on the current query object. More specifically, the query-sensitive distance measure is a weighted L_1 distance measure where the weights automatically adjust to each query.

An important property of query-sensitive embeddings is that they combine the efficiency of measuring L_1 distances with the ability to capture nonmetric structure present in the original space. In Section 3 we prove that for a wide range of finite nonmetric spaces, we can construct isometric query-sensitive embeddings. Such isometric embeddings can be achieved because a query-sensitive L_1 distance measure is not constrained to be metric, and in particular, is not constrained to be symmetric, or to obey the triangle inequality.

Regardless of whether the original space is metric or nonmetric, a query-sensitive distance measure can improve embedding quality by identifying, for each query object, those embedding dimensions that are the most informative for comparing this object with database objects. Identifying the most informative dimensions is an important issue that arises when objects are represented as high-dimensional vectors [Aggarwal 2001], and addressing this issue can help in defining meaningful high-dimensional distance measures. As demonstrated in the experimental results, using a well-chosen set of dimensions for each query is more effective than simply using all dimensions for all queries and assigning a fixed weight to each dimension.

In the experiments, we compare query-sensitive embeddings to several alternative methods for efficient nearest, neighbor retrieval. In particular, we compare our method to the original, query-insensitive BoostMap algorithm [Athitsos et al. 2004], as well as to FastMap [Faloutsos and Lin 1995] and VP-trees [Yianilos 1993]. Experiments are performed on two datasets: the MNIST database of handwritten digits [LeCun et al. 1998] with shape-context matching [Belongie et al. 2002] as the underlying distance measure, and a time-series database [Vlachos et al. 2003] with constrained dynamic time warping as the underlying distance measure. In both datasets, query-sensitive embeddings yield superior performance with respect to the original BoostMap method, FastMap, and VP-trees. For a fixed budget of exact distance computations per query and for different integers k, the new method correctly retrieves all k nearest neighbors for a significantly higher fraction of queries. In additional experiments, we construct query-sensitive embeddings and traditional, query-insensitive embeddings by minimizing embedding stress. In both our datasets, query-sensitive embeddings achieve a smaller stress value on distances between database objects and previously unseen queries.

2. RELATED WORK AND BACKGROUND

In this section, we briefly survey existing methods for efficient nearest-neighbor retrieval in high-dimensional and non-Euclidean spaces. The reader can refer to Böhm et al. [2001], Hjaltason and Samet [2003b], and White and Jain [1996] for comprehensive reviews of existing nearest-neighbor methods. We also provide some background on existing embedding methods; for a good introduction

to embedding based nearest-neighbor methods, we recommend Hjaltason and Samet [2003a].

2.1 Related Work

A large amount of literature has addressed the topic of efficient nearest-neighbor retrieval. Many methods explicitly target Euclidean and vector spaces, such as Chakrabarti and Mehrotra [2000], Sakurai et al. [2000], Weber et al. [1998], Gionis et al. [1999], Li et al. [2002], Egecioglu and Ferhatosmanoglu [2000], Kanth et al. [1998], Weber and Böhm [2000], and Koudas et al. [2004]. Such methods are not applicable in non-Euclidean spaces with computationally expensive distance measures, which is a main focus of this article. One family of methods that can be applied in such spaces are tree-based methods [Yianilos 1993; Bozkaya and Özsoyoglu 1999; Traina et al. 2000; Zezula et al. 1998]. In VP-trees [Yianilos 1993], the distance between the query object and selected database objects called *pivot points*, is used to prune-out partitions of the database on the basis of the triangle inequality. This idea is extended in Bozkaya and Özsoyoglu [1999] to multiple vantage-point trees. Both methods are exact, that is, they guarantee retrieval of the true nearest-neighbor(s), provided that the underlying distance measure is metric. An approximate method using M-trees is described in Zezula et al. [1998].

The methods in Bozkaya and Özsoyoglu [1999], Yianilos [1993], and Zezula et al. [1998] are based on metric properties. Designing general indexing methods for non-metric spaces is a much harder task, and various techniques use domain-specific properties in order to provide efficient indexing for specific domains. Several methods address the problem of robust evaluation of similarity queries on time-series databases when using nonmetric distance functions [Keogh 2002; Vlachos et al. 2003; Yi et al. 1998]. These techniques use the filter-and-refine approach, where a computationally efficient approximation of the original distance is utilized in the filtering step. Query speedup is achieved by pruning a large part of the search space at the filter step. Then, the original, accurate, but more expensive distance measure is applied to the few remaining candidates during the refinement step. In our experimental evaluation we compare our approach with the technique presented in Vlachos et al. [2003]. Also related to our setting is work on distance-based indexing for string similarity. In Sahinalp et al. [2003], special modifications to distance-based indices [Bozkaya and Özsoyoglu 1999; Traina et al. 2000; Yianilos 1993] are proposed for indexing distance functions which are *almost* metric, meaning that they satisfy the triangle inequality up to a constant. Pruning criteria that are based on the triangle inequality are adapted in Sahinalp et al. [2003] to work with the relaxed version of the triangle inequality.

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 spaces into a real vector space [Athitsos et al. 2004; Bourgain 1985; Faloutsos and Lin 1995; Hristescu and Farach-Colton 1999; Roweis and

Saul 2000; Tenenbaum et al. 2000; Wang et al. 2000; Young and Hamer 1987]. Some of these methods, in particular MDS [Young and Hamer 1987], Bourgain embeddings [Bourgain 1985; Hjaltason and Samet 2003a], LLE [Roweis and Saul 2000], and Isomap [Tenenbaum et al. 2000] are not targeted at speeding-up online similarity retrieval because they still need to evaluate exact distances between the query and most or all database objects. Online queries can be efficiently handled by Lipschitz embeddings [Hjaltason and Samet 2003a], FastMap [Faloutsos and Lin 1995], MetricMap [Wang et al. 2000], SparseMap [Hristescu and Farach-Colton 1999], and BoostMap [Athitsos et al. 2004].

In Lipschitz embeddings [Hjaltason and Samet 2003a], each coordinate of an object's embedding is set to the smallest distance between this object and a subset of database objects. Bourgain embeddings [Bourgain 1985] are a special case of Lipschitz embeddings. While Bourgain embeddings are not designed for nearest-neighbor retrieval, SparseMap [Hristescu and Farach-Colton 1999] provides an approximation of Bourgain embeddings that is explicitly designed for efficient retrieval. FastMap [Faloutsos and Lin 1995] and MetricMap [Wang et al. 2000] define embeddings using formulas that, in Euclidean spaces, approximately correspond to PCA or SVD. BoostMap [Athitsos et al. 2004] uses as building blocks simple, one-dimensional (1D) embeddings, and combines them in an optimized high-dimensional embedding using machine learning.

Existing embedding methods typically assume that the original space has either Euclidean [Faloutsos and Lin 1995; Wang et al. 2000] or metric properties [Hristescu and Farach-Colton 1999]. While these methods can be applied to arbitrary spaces, applying such methods to spaces that violate those underlying assumptions is inherently heuristic. The BoostMap method [Athitsos et al. 2004] explicitly maximizes the amount of similarity structure preserved by the embedding, and the optimization algorithm does not rely on any metric properties. Nevertheless, like the methods in Faloutsos and Lin [1995], Hristescu and Farach-Colton [1999], and Wang et al. [2000], BoostMap maps objects into a metric L_p space. When the original space is nonmetric, an embedding to a metric space cannot preserve any non-metric structure, such as violations of symmetry or the triangle inequality. Query-sensitive embeddings overcome this limitation because the distance measure used for comparing vectors is not constrained to be metric.

Another important limitation of existing embedding methods is that they compare vectors using global L_p distance measures, where each dimension of the embedding is assigned a fixed weight. In contrast, query-sensitive embeddings automatically adjust the importance of each dimension, depending on the objects that are being compared. Nonglobal distance measures have also been proposed in Paredes and Vidal [2000], Domeniconi et al. [2002], Hastie and Tibshirani [1996], and Hinneburg et al. [2000]. However, these methods have been designed to solve different problems than that of efficient nearest-neighbor retrieval, which we target in this article. The methods in Paredes and Vidal [2000], Domeniconi et al. [2002], and Hastie and Tibshirani [1996] optimize nearest-neighbor classification accuracy, while in Hinneburg et al. [2000], query-sensitive distances are optimized based on an unsupervised criterion of distance quality.

This article is an extended version of Athitsos et al. [2005], which introduced query-sensitive embeddings for nearest-neighbor retrieval. As in Athitsos et al. [2005], the embedding construction algorithm is an adaptation of the BoostMap algorithm [Athitsos et al. 2004] to the problem of query-sensitive embedding construction. In this article we describe for the first time some key theoretical properties of query-sensitive embeddings, including contractiveness in metric spaces, and the ability to preserve nonmetric structure of the original space. We also describe a method for significantly speeding-up the embedding construction algorithm by sampling the training set, and we introduce an algorithm for constructing query-sensitive embeddings so as to minimize embedding stress. Another related work is Athitsos, Alon, and Sclaroff [2005], where we describe how to speed-up nearest-neighbor classification using a cascade of embedding-based approximate nearest-neighbor classifiers. The cascade method can be applied on top of any embedding-based nearest-neighbor retrieval method, including the method proposed in this article, but also the those in Athitsos et al. [2004], Faloutsos and Lin [1995], Hristescu and Farach-Colton [1999], and Wang et al. [2000].

2.2 Background

We use X to denote a set of objects, and D_X to denote a distance measure defined on X. For example, X can be a set of images of handwritten digits, and D_X can be shape-context matching as defined in Belongie et al. [2002]. However, any X and D_X can be used in the formulations described in this article.

2.2.1 *Some Simple Embeddings.* An embedding $F : X \to \mathbb{R}^d$ is a function that maps any object $x \in X$ into a d-dimensional vector $F(x) \in \mathbb{R}^d$. Distances in $\mathbb{R}^d$ are measured using the Euclidean (L_2) metric, or some other L_p metric. It is assumed that measuring a single L_p distance between two vectors is significantly faster than measuring a single distance D_X between two objects of X. This assumption is obeyed in the example datasets we used in our experiments. For example, with our PC, we can measure close to a million L_1 distances between vectors in $\mathbb{R}^{100}$ in one second, whereas only 15 shape-context distances can be evaluated per second.

A simple way to define one-dimensional (1D) embeddings is using prototypes [Hjaltason and Samet 2003a]. In particular, given an object $r \in X$, we can define an embedding $F^r : X \to \mathbb{R}$ as follows:

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

We call embeddings of type F^r *reference-object embeddings*. The prototype r that is used to define F^r is typically called a *reference object* or a *vantage object* [Hjaltason and Samet 2003a]. Figure 1 illustrates an example of such an embedding.

In order to preserve more structure from the original space, we can use multiple reference objects to define a multidimensional embedding. For example, using d reference objects, we obtain a d-dimensional embedding:

$$F^{r_1,\ldots,r_d}(x) = (F^{r_1}(x), \ldots, F^{r_d}(x)). \tag{2}$$

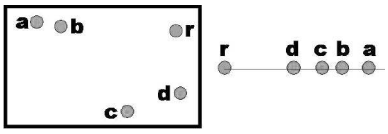

Fig. 1. An embedding F^r of five 2D points into the real line, using r as the reference object. The target of each 2D point on the line is labeled with the same letter as the 2D point.

If x_1 and x_2 are far from each other, they are typically less likely to be mapped close to each other using such a multidimensional embedding than by using a simple 1D embedding F^r.

Another family of simple 1D embeddings, for which we use the term *line-projection embeddings*, is proposed in Faloutsos and Lin [1995]. The idea is to choose two objects $x_1, x_2 \in X$, called pivot objects, and then, given an arbitrary $x \in X$, to define the embedding F^{x_1,x_2} of x to be the *projection* of x onto the "line" $\overline{x_1x_2}$:

$$F^{x_1,x_2}(x) = \frac{D_X(x,x_1)^2 + D_X(x_1,x_2)^2 - D_X(x,x_2)^2}{2D_X(x_1,x_2)}. \quad (3)$$

The reader can find in Faloutsos and Lin [1995] an intuitive geometric interpretation of this equation, based on the Pythagorean theorem. In a manner similar to Eq. (2), we can define a d-dimensional embedding by concatenating d 1D line projection embeddings. FastMap [Faloutsos and Lin 1995] is a more elaborate method for creating a multidimensional embedding using line-projection embeddings as building blocks.

2.2.2 *Filter-and-Refine Retrieval.* In applications where we are interested in retrieving the k nearest neighbors for a query object q, a d-dimensional embedding F can be used in a filter-and-refine framework [Hjaltason and Samet 2003a], as follows: First, we perform an offline preprocessing step in which we compute and store vector $F(x)$ for every database object x. Then, given a previously unseen query object q, we perform the following three steps:

—Embedding step: Compute $F(q)$ by measuring the distances between q and the reference objects and/or pivot objects used to define F.
—Filter step: Find the database objects whose associated vectors are the p most similar vectors to $F(q)$.
—Refine step: Sort these p candidates by evaluating the exact distance D_X between q and each candidate.

The assumption is that the distance measure D_X is computationally expensive and that evaluating distances between vectors is much faster. The filter step discards most database objects by measuring distances between vectors. The refine step applies D_X only to the top p candidates. The best choice of parameters p and d (embedding dimensionality) will depend on domain-specific parameters like k (i.e., how many of the nearest neighbors of an object we want to retrieve), the time it takes to compute the distance D_X, the time it takes to compare d-dimensional vectors, and the desired retrieval

accuracy (i.e., how often we are willing to miss some of the true k nearest neighbors).

3. MOTIVATION FOR QUERY-SENSITIVE DISTANCE MEASURES

There exist several methods for constructing high-dimensional embeddings, for example, FastMap [Faloutsos and Lin 1995], BoostMap [Athitsos et al. 2004], or simply using Eq. (2) with randomly chosen reference objects. Each method uses a different technique in order to capture as much as possible of the structure of the original space X. However, existing methods have limited themselves to the task of assigning a vector of coordinates to each object. These vectors are often compared using the standard Euclidean distance without weights [Faloutsos and Lin 1995; Hristescu and Farach-Colton 1999]. BoostMap uses an L_1 metric and assigns weights to each dimension, but its representational power is still equivalent to that of a method using an unweighted distance: Any weighted L_1 metric can be isometrically converted to an unweighted L_1 by simple scaling of each dimension.

Query-sensitive embeddings are embeddings into a vector space with a query-sensitive distance measure, namely, a measure where the weight of each dimension depends on the query. Using a query-sensitive distance measure enhances the modeling power of embeddings and overcomes certain limitations of standard embedding methods that use global distance measures. For example, while any finite metric space can be isometrically embedded into an L_∞ space, nonmetric spaces clearly cannot be isometrically embedded into metric L_p spaces, since isometry preserves violations of metric properties. It is also well-known that some finite metric spaces X cannot be embedded to a Euclidean (L_2) space with better than $O(\log |X|)$ distortion, where $|X|$ is the number of objects in X [Hjaltason and Samet 2003a]. Using query-sensitive embeddings, on the other hand, we can isometrically embed into a real vector space any finite metric space, and any finite nonmetric space with a reflexive distance measure.

PROPOSITION 3.1. *For any finite space X with a reflexive distance measure D_X, there exists a query-sensitive isometric embedding to $\mathbb{R}^{|X|}$, that is, the real vector space of dimension $|X|$.*

PROOF. Since X is finite, it can be represented as $X = \{x_1, \ldots, x_{|X|}\}$ so that for any i, k, $x_i = x_k$ iff $i = k$. Then, since D_X is reflexive, it follows that $D_X(x_i, x_k) = 0$ iff $i = k$.

We will explicitly construct an isometric embedding F and the associated query-sensitive distance measure D. By definition, F is isometric if the following holds: For any $x_i, x_j \in X$, $D_X(x_i, x_j) = D(F(x_i), F(x_j))$. We define F as follows:

$$F(x) = (D_X(x_1, x), D_X(x_2, x), \ldots, D_X(x_{|X|}, x)) . \tag{4}$$

Now, we define an auxiliary function $S(y) : \mathbb{R} \rightarrow \{0, 1\}$:

$$S(y) = \begin{cases} 1 \text{ if } y = 0 . \\ 0 \text{ otherwise} . \end{cases} \tag{5}$$

Finally, if $u, v \in \mathbb{R}^{|X|}$, $u = (u_1, \ldots, u_{|X|})$ and $v = (v_1, \ldots, v_{|X|})$, we define distance measure $D(u, v)$:

$$D(u, v) = \sum_{k=1}^{|X|} (S(u_k)|u_k - v_k|) . \quad (6)$$

Since reflexivity holds, $D_X(x_i, x_k) = 0$ iff $i = k$. Therefore, the kth coordinate of $F(x_i)$, which is equal to $D_X(x_i, x_k)$, is zero iff $i = k$. Therefore, it is trivial to verify that $D(F(x_i), F(x_j)) = D_X(x_i, x_j)$. □

Distance measure D is query-sensitive: Assuming that vector u in Eq. (6) is the embedding $F(q)$ of a query object $q \in X$, the weight assigned to the kth dimension depends on whether the embedding of the query has a zero or nonzero value at this dimension. Clearly, the isometric construction we have described is not really useful in the context of nearest-neighbor retrieval, where typically, we cannot assume that the query object is identical to a database object, or to a reference object used in the embedding. Furthermore, the proof entails a certain amount of "cheating," since our construction essentially boils down to defining, for each query x_i, a custom-made embedding that uses x_i as a reference object. However, the mere fact that query-sensitive embeddings allow this kind of "cheating" illustrates the additional modeling power we gain by query-sensitivity: No such trick can be used with a query-insensitive L_p metric.

At the same time, Proposition 3.1 can be used to demonstrate an important property of query-sensitive embeddings: the ability to capture nonmetric structure. The only metric property used in the proof of Proposition 3.1 is reflexivity. Symmetry and the triangle inequality were not used, and therefore the proof applies also to any space with a reflexive nonmetric distance measure. The fact that we can isometrically embed nonmetric spaces into a vector space with a query-sensitive distance measure demonstrates that such query-sensitive distance measures are not constrained to be metric. Therefore, embeddings with query-sensitive distance measures can map objects into vectors, while preserving violations of metric properties such as symmetry or the triangle inequality.

The ability to capture nonmetric structure is an important advantage of query-sensitive embeddings. Other existing embedding methods typically map objects into a vector space with a global L_p metric, and thus are inherently unable to preserve non-metric structure. At the same time, nonmetric distance measures are frequently used in many domains, including pattern recognition and data mining. Examples of nonmetric distance measures are shape-context matching [Belongie et al. 2002], dynamic time warping [Kruskall and Liberman 1983], the Chamfer distance [Barrow et al. 1977], and the k-median Hausdorff distance [Huttenlocher et al. 1993]. Such nonmetric measures often agree with human perceptions of similarity. For example, given pictures of a man, a horse, and a centaur (Figure 2), humans tend to find the centaur similar to both the man and the horse, whereas the man and the horse are rated as dissimilar [Jacobs et al. 2000]. The k-median Hausdorff distance reproduces this result by providing a small distance for the centaur-man and centaur-horse pairs, and

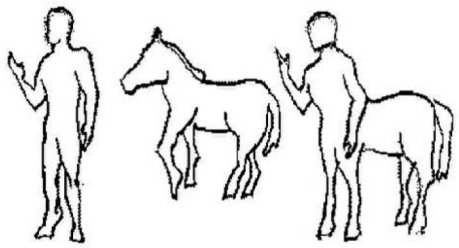

Fig. 2. An example (reprinted with permission from Jacobs et al. [2000]) of nonmetric similarity structure. Under both human notions of similarity and the k-median Hausdorff distance, the centaur is considered to be very similar to both the man and the horse, whereas the man and the horse are considered to be dissimilar.

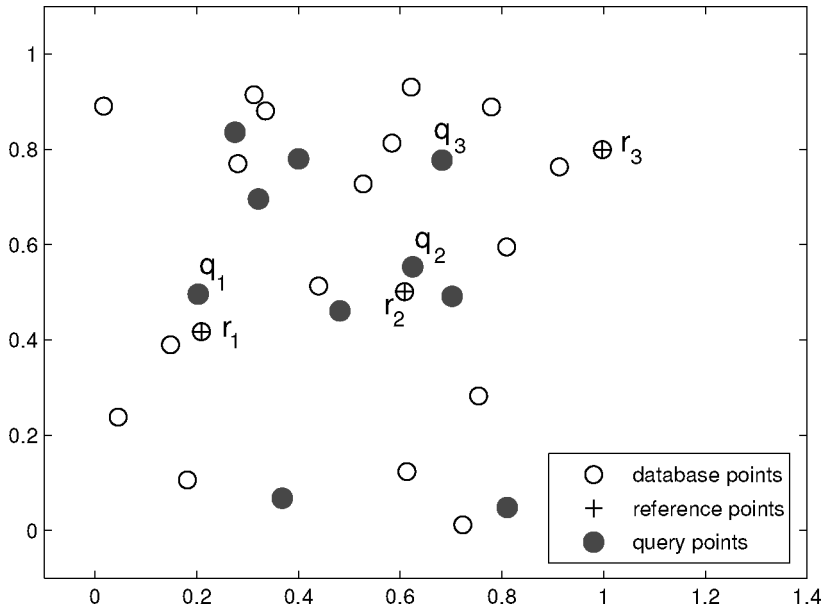

Fig. 3. A toy example illustrating the use of query-sensitive embeddings. The original space is the set of points in the plane. We define a 3D embedding using reference objects r_1, r_2, r_3. As explained in the text, assigning query-sensitive weights to each embedding dimension would improve the accuracy of the embedding for queries q_1, q_2, q_3.

a large distance (greater than the sum of the centaur-man and centaur-horse distances) for the man-horse pair. Mapping such objects into a metric space would fail to preserve this structure by forcing the distance between the man and the horse not to be larger than the sum of distances between the centaur and the man and between the man and the horse. Query-sensitive embeddings, on the other hand, can naturally preserve this type of nonmetric structure.

Another property of query-sensitive embeddings is that they can assign, given a query object, different weights to different dimensions, favoring dimensions that are more informative for this query object. This property can be useful, regardless of whether the original space is metric or nonmetric. Figure 3

illustrates a quantitative example. In this toy example, we define a three-dimensional embedding F of the plane. There are twenty database objects, three of which (indicated as r_1, r_2, r_3) are selected as reference objects. Using these reference objects, we define embedding $F(x) = (F^{r_1}(x), F^{r_2}(x), F^{r_3}(x))$, and we use the L_1 distance to compare the embeddings of two objects. There are ten query objects, three of which are marked as q_1, q_2, q_3, and there are 3,800 triples (q, a, b) we can form by picking q from the query objects, and the pair a, b from the database objects.

To evaluate embedding quality, we consider, as in Athitsos et al. [2004], that F *fails* on triple (q, a, b) if either: (a) q is closer to a than to b and $F(q)$ is closer to $F(b)$ than to $F(a)$ or (b) q is closer to b than to a and $F(q)$ is closer to $F(a)$ than to $F(b)$. Embedding F fails on 23.5% of the the 3,800 triples. In contrast, the 1D embeddings F^{r_1}, F^{r_2}, F^{r_3} perform worse than F, since they fail, respectively, on 39.2%, 36.4%, and 26.6% of the triples. However, if we restrict our attention to triples (q, a, b) where $q = q_1$, then F^{r_1} does better than F: F^{r_1} fails on 5.8% of these triples, whereas F fails on 11.6% of them. Similarly, for $q = q_2$ and $q = q_3$, respectively, F^{r_2} and F^{r_3} are more accurate than F. Therefore, for query objects q_1, q_2, q_3, it would be beneficial to use a query-sensitive weighted L_1 measure that would, respectively, use only the first, second, and third dimension of F.

Overall, we have seen that query-sensitive distance measures provide us with the modeling power to preserve nonmetric structure, and to capture the fact that different dimensions are more important for different queries.

4. CONSTRUCTING A QUERY-SENSITIVE EMBEDDING

We now describe a method for constructing query-sensitive embeddings for the purposes of approximate nearest-neighbor retrieval. The method is based on the BoostMap algorithm [Athitsos et al. 2004; Athitsos 2006], and modifies that algorithm in order to produce a query-sensitive distance measure.

4.1 Associating Embeddings with Classifiers

Any F acts as a classifier for the following binary classification problem: Given three objects $q, a, b \in X$, is q closer to a or to b? If we know F, but do not know the exact distances $D_X(q, a)$ and $D_X(q, b)$, we can provide an answer by simply checking if $F(q)$ is closer to $F(a)$ or to $F(b)$. If the answer we obtain using F is wrong, we say that F fails on triple (q, a, b) [Athitsos 2006].

Let $k_{\max}$ be the maximum number of nearest neighbors we may want to retrieve for any query object. Our goal then is to construct an embedding F that preserves the $k_{\max}$-nearest-neighbor structure of X as well as possible. Let $U \subset X$ be the set of database objects, and let $\mathbb{T}_{k_{\max}}$ be the set of triples (q, a, b) such that $q \in X$, $a, b \in U$, and a is a k-nearest neighbor of q. If F never fails on such triples, then F maps the $k_{\max}$-nearest neighbors of q in U to the $k_{\max}$-nearest neighbors of $F(q)$ in $F(U)$, and thus F perfectly preserves the $k_{\max}$-nearest-neighbor structure. The fraction of triples in $\mathbb{T}_{k_{\max}}$ on which F does not fail is a quantitative measure of how well F preserves the $k_{\max}$-nearest-neighbor structure [Athitsos et al. 2005; Athitsos 2006]. Consequently, we want to construct F in a way that minimizes its failure rate on the set $\mathbb{T}_{k_{\max}}$.

Simple 1D embeddings that are defined, for example, using reference objects or line projections, are expected to act as *weak classifiers* [Athitsos 2006; Schapire and Singer 1999], that is, they will probably have a high error rate, but at the same time they should provide answers that are, on average, more accurate than a random guess, which would have an error rate of 50%. The key insight in the BoostMap algorithm [Athitsos et al. 2004; Athitsos 2006] is that by associating embeddings with classifiers, we can reduce the problem of embedding construction to the problem of combining many weak classifiers into a strong classifier. A well-known and widely used solution to the latter problem is the AdaBoost algorithm [Schapire and Singer 1999]. The BoostMap algorithm uses AdaBoost to construct a high-dimensional embedding out of 1D embeddings of type F^r and F^{x_1,x_2}.

At a high level, our embedding construction method is an adaptation of the BoostMap algorithm and consists of the following steps:

(1) We start by specifying a large family of 1D embeddings, using well-known definitions from prior embedding methods.
(2) We use 1D embeddings to define binary classifiers which estimate for object triples (q, a, b) if q is closer to a or to b.
(3) We use AdaBoost to combine many classifiers into a single classifier H, which we expect to be significantly more accurate than the simple classifiers associated with 1D embeddings.
(4) We use H to define a d-dimensional embedding F_{out}, and a query-sensitive weighted L_1 distance measure D_{out}.

In order to produce a query-sensitive embedding, the adapted BoostMap algorithm described in this section differs from the original BoostMap algorithm of Athitsos et al. [2004] and Athitsos [2006] in Steps 2,3, and 4, as explained in the remainder of this section.

4.2 Defining Query-Sensitive Classifiers from 1D Embeddings

As described earlier, every embedding F corresponds to a classifier that classifies triples (q, a, b) of objects in X. Formally, we can say that a triple (q, a, b) is of type 1 if q is closer to a than to b, type 0 if q is equally close to a and to b, and type -1 if q is closer to b than to a. Given an embedding F and a distance measure D for comparing vectors, we define the classifier $\tilde{F}$ associated with embedding F as:

$$\tilde{F}(q, a, b) = D(F(q), F(b)) - D(F(q), F(a)) . \tag{7}$$

The sign of $\tilde{F}(q, a, b)$ is an estimate of whether triple (q, a, b) is of type 1, 0, or -1. We should note that if F is a 1D embedding and D is an L_p distance measure (where $p > 0$), then $\tilde{F}(q, a, b) = |F(q) - F(b)| - |F(q) - F(a)|$.

Sometimes, $\tilde{F}$ may do a really good job on triples (q, a, b) when q is in a specific region, but at the same time, it may be beneficial to ignore $\tilde{F}$ when q is outside this region. For example, suppose that we have an embedding F^r defined using reference object r. If $q = r$, then $\tilde{F}^r$ will classify correctly all

triples (q, a, b), where a and b are any two objects of space X. If $q \neq r$, we still expect that, the closer q is to r, the more accurate $\tilde{F}^r$ will be on triples (q, a, b), as illustrated in Figure 3.

In Athitsos et al. [2004], the weak classifiers used by AdaBoost are of type $\tilde{F}$, with F being a 1D embedding. We propose to use a different type of classifier, which can explicitly model the fact that a 1D embedding F can be more useful in some regions of the space and less useful in others.

In particular, given a 1D embedding F, we need a function $S(q)$ (which we call a *splitter*) that will estimate, given a query q, whether classifier $\tilde{F}$ is useful. More formally, if X is the original space, we use the term splitter to denote any function mapping X to the binary set $\{0, 1\}$. We can readily define splitters using 1D embeddings. Given a 1D embedding $F : X \to \mathbb{R}$ and a subset $V \subset \mathbb{R}$, we can define a splitter $S_{F,V} : X \to \{0, 1\}$ and a *query-sensitive classifier* $\tilde{Q}_{F,V} : X^3 \to \mathbb{R}$ as follows:

$$S_{F,V}(q) = \begin{cases} 1 & \text{if } F(q) \in V \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

$$\tilde{Q}_{F,V}(q, a, b) = S_{F,V}(q)\tilde{F}(q, a, b). \tag{9}$$

At an intuitive level, $\tilde{F}$ is itself a classifier of triples (q, a, b). Moreover, $\tilde{Q}_{F,V}$ is a cropped version of $\tilde{F}$ that gives 0 (i.e., a neutral result) whenever $F(q) \notin V$. For example, if $F = F^r$ for some reference object r, and $V = [0, \tau]$ for some positive threshold τ, then splitter $S_{F,V}(q)$ accepts object q if it is within distance τ of r. Therefore, the query-sensitive classifier $\tilde{Q}_{F,V}$ will apply $\tilde{F}$ only if q is sufficiently close to r. By choosing τ in an appropriate way, we can capture the fact that $\tilde{F}$ should only be applied to objects within a specified distance from reference object r.

4.3 The Training Algorithm

The AdaBoost algorithm (taken, with minor modifications, from Schapire and Singer [1999]) is shown in Figure 4. AdaBoost assumes that we have a "weak learner" module, which we can call at each round to obtain a new weak classifier. The output of AdaBoost is a "strong classifier" which is a linear combination of the weak classifiers chosen at each round. The goal is to construct a linear combination that achieves much higher accuracy than the individual weak classifiers.

In our case, the AdaBoost algorithm is used to construct a classifier that operates on triples of objects. As stated in Section 4.1, our goal is to minimize the classification error rate on the set $\mathbb{T}_{k_{\max}}$ of triples (q, a, b) such that $q \in X$, $a, b \in U$ are database objects, and a is a $k_{\max}$-nearest neighbor of q. Therefore, in principle, the training triples $o_i = (q_i, a_i, b_i)$ should be sampled from $\mathbb{T}_{k_{\max}}$. However, in order to do this we need to compute distances from a large set of possible queries q to all objects in U. For efficiency reasons, we choose training triples from a subset $X_{\text{tr}} \subset U$. We choose triples (q, a, b) such that $q, a, b \in X_{\text{tr}}$, and a is a k_1-nearest neighbor of q in X_{tr}, where $k_1 = k_{\max}\frac{|X_{\text{tr}}|}{|U|}$ [Athitsos et al. 2005]. The ith training triple (q_i, a_i, b_i) is associated with a class label y_i, which is 1 if q_i is closer to a_i and -1 if q_i is closer to b_i.

Given: $(o_1, y_1), \ldots, (o_t, y_t)$; $o_i \in \mathcal{G}, y_i \in \{-1, 1\}$.
Initialize $w_{i,1} = \frac{1}{t}$, for $i = 1, \ldots, t$.
For $j = 1, \ldots, J$:

(1) Train weak learner using training weights $w_{i,j}$.
(2) Get weak classifier $h_j : \mathcal{G} \rightarrow \mathbb{R}$.
(3) Choose $\alpha_j \in \mathbb{R}$.
(4) Set training weights $w_{i,j+1}$ for the next round as follows:

$$w_{i,j+1} = \frac{w_{i,j} \exp(-\alpha_j y_i h_j(x_i))}{z_j} . \tag{10}$$

where z_j is a normalization factor (chosen so that $\sum_{i=1}^{t} w_{i,j+1} = 1$).

Output the final classifier:

$$H(x) = \sum_{j=1}^{J} \alpha_j h_j(x). \tag{11}$$

Fig. 4. The AdaBoost algorithm. This description is largely copied from Schapire and Singer [1999].

We also use the same process to choose a set of validation triples from a set $X_{\text{val}} \subset U$ of the same size as X_{tr} and disjoint from X_{tr}. Validation triples are used for deciding when to stop training, as described later.

To create a pool of 1D embeddings/weak classifiers, we need to specify a set $C \subset U$ of *candidate objects*. Elements of C will be used as reference objects and pivot objects to define 1D embeddings of type F^r and F^{x_1,x_2}. We need to precompute a matrix of distances between any two objects in C, and a matrix of distances from each $c \in C$ to each q_i, a_i, and b_i appearing in one of the training triples.

The weak classifiers considered in Steps 1 and 2 of Figure 4 are classifiers $\tilde{Q}_{F,V}$ as defined in Eq. (9), where F is some 1D embedding defined using reference objects or pivot objects from the set C of candidate objects. To pick a range V for $\tilde{Q}_{F,V}$, we simply compute the values $F(x)$ for every object appearing in a training triple (q_i, a_i, b_i), and set V to be a random interval of $\mathbb{R}$ containing some of these values. We form many such ranges V for each F, and for each range, we measure the training error, namely, the classification error of classifier $\tilde{Q}_{F,V}$, on the training triples. When we measure the training error, we weigh each training triple o_i by the current weight $w_{i,j}$ of this triple in training round j. Therefore, the error of $\tilde{Q}_{F,V}$ will be different at each training round.

At training round j, we choose (randomly) a large number of 1D embeddings. For each selected 1D embedding F, we find the range $V_{F,j}$ that achieves the lowest training error at round j. The next classifier will be chosen among the classifiers $\tilde{Q}_{F,V_{F,j}}$. The function $Z_j(\tilde{Q}, \alpha)$ gives a measure of how useful it would be to choose $h_j = \tilde{Q}$ and $\alpha_j = \alpha$ at training round j:

$$Z_j(\tilde{Q}, \alpha) = \sum_{i=1}^{t} (w_{i,j} \exp(-\alpha y_i \tilde{Q}(q_i, a_i, b_i))) . \tag{12}$$

Full details of the significance of Z_j can be found in Schapire and Singer [1999]. Here, it suffices to say that the lower $Z_j(\tilde{Q}, \alpha)$, the more beneficial it is to choose $h_j = \tilde{Q}$ and $\alpha_j = \alpha$.

Now we are ready to specify how to implement Steps 1–3 in Figure 4 for each training round $j = 1, \ldots, J$. In Step 1, we find the optimal α for each weak classifier $\tilde{Q}_{F,V_{F,j}}$. Then, in Steps 2 and 3, we set h_j and α_j, respectively, to be the weak classifier and weight that yielded the lowest overall value of Z_j. An optional setting, which we use in our experiments, is to constrain each α_j to be nonnegative. This allows the resulting embedding to be *contractive* [Hjaltason and Samet 2003a] in metric spaces, as discussed in Section 5.4.

The training algorithm stops if either of two conditions is satisfied:

—We have reached a specified maximum number of training rounds or
—the error rate of the classifier on the validation triples has not improved during a specified number of training rounds.

4.3.1 *Training Output: Embedding and Distance.* The output of the training stage is a classifier H of the following form:

$$H = \sum_{j=1}^{J} \alpha_j \tilde{Q}_{F'_j,V_j} . \tag{13}$$

Each $\tilde{Q}_{F'_j,V_j}$ is associated with a 1D embedding F'_j. Classifier H has been trained to estimate, for triples of objects (q, a, b), if q is closer to a or to b. However, our goal is to actually construct not just a classifier of triples of objects, but an embedding. Here, we discuss how to define such an embedding F_{out}, and an associated distance measure D_{out} to be used to compare vectors.

A particular 1D embedding F may have been selected at multiple training rounds, and thus can be equal to multiple F'_j's occurring in the definition of classifier H. We construct the set $\mathbb{F}$ of all unique 1D embeddings used in H as $\mathbb{F} = \bigcup_{j=1}^{J} \{F'_j\}$, and we denote the elements of $\mathbb{F}$ as $F_1, \ldots, F_d$. It is important to distinguish between notation F_i (for elements of $\mathbb{F}$), and notation F'_i (for embeddings appearing in Eq. (13)).

The embedding $F_{\text{out}} : X \rightarrow \mathbb{R}^d$ is defined as follows:

$$F_{\text{out}}(x) = (F_1(x), \ldots, F_d(x)). \tag{14}$$

Obviously, this is a d-dimensional embedding. We note here that dimensionality d is equal to the number of unique embeddings F_j, and not to J, which is the total number of training rounds performed by AdaBoost.

Before defining distance measure D_{out}, we first need to define an auxiliary function $A_i(q)$ which assigns a weight to the ith dimension, for $i = 1, \ldots, d$:

$$A_i(q) = \sum_{j:((j\in\{1,\ldots,J\})\wedge(F_i=F'_j)\wedge(F_i(q)\in V_j))} \alpha_j . \tag{15}$$

In words, given object q for dimension i, we go through all the weak classifiers $\tilde{Q}_{F'_j,V_j}$ that make up H. For each such classifier, we check if the splitter $S_{F'_j,V_j}$

accepts q (i.e., we check if $F'_j(q) \in V_j$), and we also check if $F'_j = F_i$. If these conditions are satisfied, we add the weight α_j to $A_i(q)$.

Let $F_{\text{out}}(q) = (q_1, ..., q_d)$, and let x be some other object in X, with $F_{\text{out}}(x) = (x_1, ..., x_d)$. We define distance D_{out} as follows:

$$D_{\text{out}}((q_1, ..., q_d), (x_1, ..., x_d)) = \sum_{i=1}^{d} (A_i(q)|q_i - x_i|) . \tag{16}$$

Here, $D_{\text{out}}(v_1, v_2)$ (where v_1, v_2 are d-dimensional vectors) is like a weighted L_1 measure on $\mathbb{R}^d$, but the weights depend on v_1. Therefore $D_{\text{out}}(v_1, v_2)$ is neither symmetric nor a metric. We say that $D_{\text{out}}(v_1, v_2)$ is a *query-sensitive distance measure*, since v_1 is typically the embedding of a query, and v_2 is the embedding of a database object that we want to compare to the query.

It is important to note that because of the way we defined F_{out} and D_{out}, if we apply Eq. (7) to obtain a classifier $\tilde{F}_{\text{out}}$ from F_{out} (with D set to D_{out}), then $\tilde{F}_{\text{out}} = H$ (the proof can be found in Athitsos et al. [2005]). In words, the classifier corresponding to embedding F_{out} is equal to the output of AdaBoost. This equivalence is important because it shows that the quantity optimized by the training algorithm (i.e., classification error on triples of objects) is not only a property of the classifier H constructed by AdaBoost, but also a property of the embedding F_{out}, when coupled with query-sensitive distance measure D_{out}.

5. PROPERTIES AND DISCUSSION OF THE METHOD

In this section we take a closer look at some properties of query-sensitive embeddings and the proposed algorithm for constructing such embeddings.

5.1 Complexity

The complexity analysis of the training algorithm and online retrieval is the same as for the original BoostMap algorithm, and discussed in detail in Athitsos et al. [2004;2005], and Athitsos [2006]. For the training algorithm, if at each training round we evaluate m weak classifiers by measuring their performance on t training triples, the computational time per training round is $O(mt)$. Before we even start the training algorithm, we need to compute distances D_X from every object in C (the set of objects that we use to form 1D embeddings) to every object in C and to every object in X_{tr} (the set of objects from which we form training triples). We also need all distances between pairs of objects in X_{tr}. If (as in our experiments) C and X_{tr} have an equal number of elements, then the number of distances that we need to precompute is quadratic to $|C|$. With respect to the *online* filter-and-refine retrieval cost, computing the d-dimensional embedding of a query object takes $O(d)$ time and requires $O(d)$ evaluations of D_X. Comparing the embedding of the query to the embeddings of n database objects takes time $O(dn)$.

5.2 Optimization Issues

It is important to point out that the embedding construction algorithm described in this section does not guarantee finding a globally optimal solution,

that is, a query-sensitive embedding whose classification error on triples of objects is globally optimal. The algorithm merely converges to a local optimum. This property is inherited from the original AdaBoost algorithm, which greedily selects at each training round a weak classifier and an associated weight that give optimal results when combined with previously chosen weak classifiers and weights. At the same time, AdaBoost is a very popular machine learning method that has been applied successfully in multiple domains, because of its efficiency, ability to deal with very high-dimensional data, and the empirically observed property that it resists overfitting the training data and produces classifiers that work well on data that was not used for training.

It is also important to stress that other optimization methods can also be used for the task of constructing a query-sensitive embedding. BoostMap, with its use of AdaBoost, provides a good tradeoff between efficiency of training algorithm and accuracy of the resulting embedding. A computationally intractable alternative would be to perform brute-force search over the space of all possible linear combinations of query-sensitive embeddings. A feasible alternative is to optimize the embedding using gradient descent, after providing one or more starting points for the descent. Greedily choosing 1D embeddings one-by-one is another option. In Section 6 we describe such a greedy optimization method that can be used to minimize the stress of the embedding. Minimizing stress is an optimization criterion for which the BoostMap algorithm cannot be used.

5.3 Using Sampling of Triples to Speed Up Training

The run time of the training algorithm is linear to the number of training triples. At the same time, using a large number of training triples allows the training algorithm to construct a more accurate classifier. We often observe in practice that after several training rounds, the classifier constructed by AdaBoost classifies correctly all training triples, while still makeing mistakes on validation triples. This is a case of classifier overfitting. Overfitting is a more significant issue with query-sensitive than with query-insensitive embeddings [Athitsos et al. 2004] because the former have a much larger number of degrees of freedom. Using a larger number of training triples is a straightforward approach for addressing overfitting.

A solution to the dilemma of trading training time for classifier accuracy is to use a relatively large number of training triples, but only to use a sample of these triples at each training round. In particular, training time is linear to the time it takes to evaluate Eq. (12). This equation is evaluated repeatedly at each training round j to compute the Z_j value for many different choices of an embedding F, range V, and weight α. Computing Z_j takes time linear to the number of training triples. By using only a sample of training triples to evaluate Eq. (12), we can significantly speed-up training time. At the same time, while the choice of weak classifier at each training round j may overfit the sample set of triples used at this training round, the large majority of training triples are not considered at round j, and therefore not overfitted by the weak classifier chosen at this round.

In addition, we can bias the sampling so that training triples which are misclassified by the current strong classifier are overrepresented in the sample set. Naturally, in this case, the weight of the sampled misclassified training triples needs to be normalized so as to reflect the relative weight of misclassified training triples in the entire training set. By overrepresenting misclassified triples, we make available to the training algorithm a better representation of the problematic regions where classification needs to be improved.

In our experiments, we have implemented sampling of training triples. At each training round, we constrain the sampling so that half of the sampled triples are triples misclassified by the current strong classifier. Overall, using sampling we speed-up training time by an order of magnitude over the implementation we used in Athitsos et al. [2005], while obtaining comparable classification accuracy.

5.4 Contractiveness

Contractiveness is an important property of some types of embedding. When it holds, contractiveness can be used to guarantee that filter-and-refine retrieval will always return the true k-nearest neighbors for any query [Hjaltason and Samet 2003a]. An embedding F that maps space X with distance measure D_X into a vector space with distance measure D is contractive if for any $x_1, x_2 \in X$, it holds that $D(F(x_1), F(x_2)) \leq D_X(x_1, x_2)$. As explained in Hjaltason and Samet [2003a], when an embedding is contractive, then the refine step of filter-and-refine retrieval can automatically decide how many exact distance computations it needs to perform, given a query, in order to guarantee correct results.

Query-sensitive embedding F_{out} with associated distance measure D_{out}, constructed as described in Section 4.3.1, can be made contractive by dividing $D_{\text{out}}(F_{\text{out}}(q), F_{\text{out}}(x))$ with a query-sensitive normalization term, provided that:

—D_X is metric; and
—the weights α_j chosen by the training algorithm are nonnegative. As discussed in Section 4.3, our implementation ensures that all weights α_j are nonnegative.

If F_{out} contains no line-projection embeddings, the normalization term $W(q)$ that should be used is:

$$W(q) = \sum_{i=1}^{d} A_i(q) . \tag{17}$$

PROPOSITION 5.1. *Let $q, x \in X$. Suppose that $\alpha_j \geq 0$ for all j, and suppose that D_X is metric. If all dimensions of F_{out} are reference-object embeddings, then the following property holds:*

$$\frac{1}{W(q)} D_{\text{out}}(F_{\text{out}}(q), F_{\text{out}}(x)) \leq D_X(q, x) . \tag{18}$$

PROOF. If each dimension of F_{out} is a reference-object embedding, then F_{out} can be represented as $F_{\text{out}} = (F^{r_1}, \ldots, F^{r_d})$, where d is the dimensionality

of F_{out} and r_i are reference objects. We will denote $F_{\text{out}}(q)$ as $(q_1, \ldots, q_d)$ and $F_{\text{out}}(x)$ as $(x_1, \ldots, x_d)$. First, based on the triangle inequality, we can easily see that:

$$|q_i - x_i| = |D_X(q, r_i) - D_X(x, r_i)| \leq D_X(q, x) . \tag{19}$$

Using this observation, we can complete the proof:

$$\begin{aligned}
\frac{1}{W(q)} D_{\text{out}}(F_{\text{out}}(q), F_{\text{out}}(x)) &= \frac{1}{W(q)} \sum_{i=1}^{d} (A_i(q)|q_i - x_i|) \\
&\leq \frac{1}{W(q)} \sum_{i=1}^{d} (A_i(q) D_X(q, x)) \\
&= \frac{1}{W(q)} D_X(q, x) \sum_{i=1}^{d} (A_i(q)) \\
&= \frac{1}{W(q)} D_X(q, x) W(q) \\
&= D_X(q, x) . \qquad \square
\end{aligned}$$

If F_i, the ith dimension of F_{out}, is a line-projection embedding, then it is shown in Hjaltason and Samet [2003a] that $|F_i(q) - F_i(x)| \leq 3D_X(q, x)$. Therefore, if we divide $D_{\text{out}}(q, x)$ by $3W(q)$, then F_{out} is contractive, even in the case where some of its dimensions are line-projection embeddings.

We should point out that the preceding proof of contractiveness follows the same pattern as the proof in Athitsos [2006] that query-insensitive BoostMap embeddings are contractive. The key difference here is that since the distance measure is query-sensitive, the normalization factor $W(q)$ is a function of the query q, whereas for query-insensitive BoostMap embeddings, the normalization factor is a constant.

5.5 Replacing Binary Splitters with More General Functions

We have obtained query-sensitive embeddings by replacing the query-insensitive weak classifiers $\tilde{F}(q, a, b)$ used in the original BoostMap algorithm [Athitsos et al. 2004] with query-sensitive weak classifiers of the form $S(q)\tilde{F}(q, a, b)$, where S maps X to the binary set $\{0, 1\}$. However, using a binary S is essentially an implementation choice. In general, function S can be an arbitrary function mapping X to $[0, \infty)$. For example, S can be a smoothed-out version of a binary function that makes smooth transitions between regions of X mapped to 0 and regions of X mapped to 1. The training algorithm can easily be modified to choose such more complicated functions S. At the same time, optimizing a nonbinary function S may be more challenging, since it may involve searching over a larger space of parameters.

We should note that two key properties of query-sensitive embeddings constructed using binary splitters S also hold if we replace such splitters with arbitrary functions S. The first property is that of the equivalence between the classifier constructed by AdaBoost and the corresponding embedding, in other words, the fact that $\tilde{F}_{\text{out}} = H$. The second property is contractiveness. Overall,

replacing binary splitters with more general functions provides additional flexibility to the embedding optimization algorithm, and is an interesting topic for future investigation.

5.6 Dynamic Datasets

In our discussion so far, we have assumed that the database is static. In some applications, however, we may need to add or remove objects online. As long as the underlying distribution of database objects is not altered, adding and removing objects is pretty straightforward. When adding an object x, we need to compute its embedding $F_{\text{out}}(x)$. If F_{out} is d-dimensional, computing $F_{\text{out}}(x)$ requires computing, at most, $2d$ distances D_X between x and database objects.

If the underlying distribution of database objects changes significantly because of additions and removals, we may have to create a new embedding. A way to check whether the distribution of database objects has changed significantly is by measuring, at regular intervals, the error of the current embedding F_{out}, namely, the classification error of $\tilde{F}_{\text{out}}$ on triples of objects picked (from the current database distribution) in the same way we would choose training triples. When that error increases above some threshold, we can reuse the training algorithm to construct a new embedding.

6. OPTIMIZING QUERY-SENSITIVE EMBEDDINGS FOR STRESS

In Section 4 we have described a modified version of the BoostMap algorithm, which can be used for optimizing a query-sensitive embedding. The optimization criterion used (error rate on triples of objects) is a measure of the amount of nearest-neighbor structure preserved by the embedding. We should note that BoostMap and AdaBoost are applicable because we have chosen an optimization criterion that corresponds to the error rate of a binary classifier. At the same time, query-sensitive embeddings can also be defined in applications where the goal is not nearest-neighbor retrieval. For example, some applications may require optimizing the approximation of the actual pairwise distances between objects in the original space. In this case, other optimization criteria are applicable, such as minimizing the stress or distortion of the embedding. BoostMap has not been designed to work with such optimization criteria, but gradient descent and greedy optimization methods can still be applied.

In this section, we describe a greedy algorithm for constructing a query-sensitive embedding in a way that minimizes the stress of the embedding. Greedy algorithms for minimizing embedding stress have been previously proposed in the literature [Athitsos and Sclaroff 2003; Hristescu and Farach-Colton 1999] for query-insensitive embeddings. We first describe an algorithm for query-insensitive embeddings and then we propose a modified version of this algorithm that can generate query-sensitive embeddings.

6.1 Minimizing Stress for Query-Insensitive Embeddings

Stress is a measure of how well an embedding preserves distances. Given a space X with a distance measure D_X, a d-dimensional embedding $F : X \rightarrow$

$\mathbb{R}^d$, and a distance measure D on $\mathbb{R}^d$, we define the stress $\sigma(F)$ of F as follows:

$$\sigma(F) = \min_{c>0} \frac{\sqrt{E_{x_1,x_2 \in X}((cD(F(x_1), F(x_2)) - D_X(x_1, x_2))^2)}}{E_{x_1,x_2 \in X} D_X(x_1, x_2)}, \tag{20}$$

where E denotes the expected value over a set. As mentioned in Hjaltason and Samet [2003a], alternative definitions of stress are also commonly used in the literature. By minimizing over all positive scalars c, we ensure that distance measure D is optimally scaled.

Suppose that we have a pool $\mathbb{F}$ of 1D embeddings, and we want to combine some of these embeddings into a multidimensional embedding $F = (F_1, ..., F_d)$ in a way that minimizes stress. Obviously, brute-force search for the globally optimal combination is computationally prohibitive. On the other hand, it is easy to implement a greedy algorithm which selects 1D embeddings one-by-one, choosing at each step that 1D embedding which, combined with the already chosen embeddings, yields the smallest stress value [Athitsos and Sclaroff 2003; Hristescu and Farach-Colton 1999]. The algorithm proceeds as follows:

(1) $j = 1$.
(2) $F_1 = \text{argmin}_{F \in \mathbb{F}}[\sigma(F)]$.
(3) $F_{j+1} = \text{argmin}_{F \in \mathbb{F}}[\sigma((F_1, \ldots, F_j, F))]$.
(4) $j = j + 1$.
(5) Unless some stopping criterion is satisfied, go to Step 3.
(6) Output embedding $F_{\text{out}} = (F_1, \ldots, F_j)$.

The stopping criterion may be simply whether we have reached a certain maximum number $j_{\max}$ of iterations. The distance measure D used for comparing vectors can be arbitrary. In our implementation, we use an unweighted L_1 distance measure.

6.2 Minimizing Stress for Query-Sensitive Embeddings

The algorithm described before can be modified to produce a query-sensitive embedding, namely, an embedding that uses a query-sensitive distance measure D. The key modification is that at each step j, instead of simply choosing an optimal F_j, we choose an optimal combination of an F_j and an area of influence V_j for F_j. This area of influence specifies that dimension j should only be used for queries q such that $F_j(q) \in V_j$. When $F_j(q) \notin V_j$, then the query-sensitive weight for dimension j should be set to zero. We say that an object x belongs to an area of influence V_j if $F_j(x) \in V_j$.

Given embeddings $F_1, \ldots, F_d$ and corresponding areas of influence $V_1, \ldots, V_d$, a useful quantity to define for each object $x \in X$ is $\Phi(x)$, which counts the number of areas of influence V_j that object x belongs to:

$$\Phi(x) = \sum_{i=1}^{j} S_{F_i,V_i}(x). \tag{21}$$

In the preceding equation, we use the splitter function $S_{F,V}$ defined in Eq. (8) of Section 4.2. Splitter $S_{F,V}(x)$ outputs 1 if $F(x) \in V$ and 0, otherwise.

As the greedy algorithm picks d different embeddings F_j and areas of influence V_j, it may be that $\Phi(x)$ is relatively large for some objects x and relatively small for other objects. Overall, the expected average distance between the embedding of a query q and embeddings of database objects tends to increase with $\Phi(q)$. In the algorithm presented in Section 4, which minimizes classification error on triples (q, x_1, x_2), the scale of the average distance between the embedding of q and the embeddings of database objects did not matter. However, if we want to minimize stress, the scale of distances does matter because in order to measure stress, we directly compare distances D in the vector space with distances D_X in the original space. In order to make the average distance between a query q and database objects independent of $\Phi(q)$, we need to normalize all distances in the vector space by dividing them by $\Phi(q)$.

We now proceed to formally define the query-sensitive distance measure $D_{\mathcal{V}}$ to be used for embedding $F = (F_1, \ldots, F_d)$, given a sequence $\mathcal{V} = (V_1, \ldots, V_d)$ of areas of influence V_j for each F_j. For objects $x_1, x_2 \in X$, we define $D_{\mathcal{V}}(F(x_1), F(x_2))$ as follows:

$$D_{\mathcal{V}}(F(x_1), F(x_2)) = \frac{\sum_{i=1}^{d}(S_{F_i,V_i}(x_1)|F_i(x_1) - F_i(x_2)|)}{\Phi(x_1)}. \tag{22}$$

In order to select an area of influence V_j for embedding F_j, we need to search over different possible areas of influence V. In our implementation, we constrain embeddings F_j to be 1D embeddings defined using reference objects (Eq. (1)). Given an embedding F_j and a set of n objects $x_1, \ldots, x_n$ that we use to measure stress, we define a set of m candidate areas of influence $\mathbb{V}_{F_j} = \{V_{j,1}, \ldots, V_{j,m}\}$. Each $V_{j,i}$ is set to $[0, t_i]$, where t_i is set in such a way that only $\frac{mi}{m}$ objects among $x_1, \ldots, x_n$ belong to area of influence $[0, t_i]$.

Before we specify each step of the optimization algorithm, we need to introduce some additional notation. Let $\mathcal{V}$ be a sequence $(V_1, \ldots, V_d)$ of areas of influence. We denote by $\mathcal{V} \oplus V$ the sequence we obtain by appending V to the end of $\mathcal{V}$, so that $\mathcal{V} \oplus V = (V_1, \ldots, V_d, V)$. Also, given an embedding $F = (F_1, \ldots, F_d)$ and a sequence $\mathcal{V} = (V_1, \ldots, V_d)$, we define the stress $\sigma(F, \mathcal{V})$ corresponding to using F with the areas of influence specified in $\mathcal{V}$ as follows:

$$\sigma(F, \mathcal{V}) = \min_{c>0} \frac{\sqrt{E_{x_1,x_2 \in X}((cD_{\mathcal{V}}(F(x_1), F(x_2)) - D_X(x_1, x_2))^2)}}{E_{x_1,x_2 \in X}(D_X(x_1, x_2))}. \tag{23}$$

We can now describe step-by-step the greedy algorithm for constructing a query-sensitive embedding in a way that minimizes stress:

(1) $j = 1$.
(2) $\mathcal{V} = \emptyset$.
(3) $[F_1, V_1] = \text{argmin}_{F \in \mathbb{F}, V \in \mathbb{V}_F}[\sigma(F, \mathcal{V} \oplus V)]$.
(4) $\mathcal{V} = (V_1)$.
(5) $[F_{j+1}, V_{j+1}] = \text{argmin}_{F \in \mathbb{F}, V \in \mathbb{V}_F}[\sigma((F_1, \ldots, F_j, F), \mathcal{V} \oplus V)]$.
(6) $\mathcal{V} = \mathcal{V} \oplus V_{j+1}$.

(7) $j = j + 1$.
(8) Unless some stopping criterion is satisfied, go to Step 5.
(9) Output embedding $F_{\text{out}} = (F_1, \ldots, F_j)$, areas of influence $(V_1, \ldots, V_j)$.

In the experiments, we will show that in our datasets, query-sensitive embeddings constructed using this algorithm achieve smaller stress than the query-insensitive embeddings constructed using the greedy algorithm described in Section 6.1. We should note that similar greedy algorithms can also be implemented for minimizing embedding distortion, instead of stress.

7. EXPERIMENTS

We have evaluated the proposed method on two different datasets: the MNIST dataset of handwritten digits [LeCun et al. 1998] (with the shape-context distance [Belongie et al. 2002] as the exact distance measure), and a time series database [Vlachos et al. 2003] (with constrained dynamic time warping [Vlachos et al. 2003] as the exact distance measure). On each dataset, we provide two sets of results. The first set of results explicitly evaluates the benefits of using a query-sensitive distance measure by comparing our method to the original BoostMap algorithm. The second set of results compares our method both to alternative embedding methods and to VP-trees [Yianilos 1993].

7.1 Evaluation Methodology

To compare different embedding methods, we used each of them to build embeddings of various dimensions (the dimensionality ranged from 1 to 600). For each embedding method, for each k and accuracy percentage B, we report the number of exact distance computations needed in order to successfully retrieve *all* k true nearest neighbors for a percentage of query objects equal to B, while minimizing the total number of exact distance computations per query object.

In both datasets, we found that retrieval time, using any of the indexing methods we have tried, was dominated by the number of exact distance computations we needed to evaluate for each query object. In the experiments, we observed that disk access time and high-dimensional vector comparisons performed at the filter step were negligible in comparison to the time spent evaluating exact distances. For this reason, we measure retrieval efficiency in units of exact distance computations.

For each embedding method, number k of nearest neighbors, and desired accuracy rate B, we need to choose the dimensionality d of the embedding. We do this using a validation set of queries which is a subset of the database and disjoint from the test set of queries that we use to evaluate performance. On this validation set we find the value of d that minimizes the number of exact distance computations per query, given k and B. Given k, d, and B, we set parameter p of filter-and-refine retrieval to the smallest value that is required to obtain accuracy B. After choosing p, the number of exact distance computations per query object is fully specified.

After we choose a value for d, we use the d-dimensional embedding as part of filter-and-refine retrieval in order to retrieve the k-nearest neighbors of the

queries in the test set. Then, we reset p to a value that leads to retrieval accuracy B on the test set of queries, and we report the number of exact distance computations per query that correspond to d and the new choice of p. Setting p so as to obtain a desired retrieval accuracy rate allows us to directly compare the number of exact distances required by different embedding methods to obtain this accuracy rate.

7.2 Datasets

The MNIST dataset contains images of isolated handwritten digits (numbers from 0 to 9). MNIST consists of a training set of 60,000 images, which we used as the database, and a test set (disjoint from the training set) of 10,000 images that we used as query objects. Shape-context matching is a distance measure described in Belongie et al. [2002]. To compute this distance, 100 shape-context features are extracted from each image. Two images are aligned via bipartite matching between their features, which involves the computationally expensive Hungarian algorithm. The final distance is a weighted sum of three terms: the cost of matching shape-context features, the cost of alignment, and the intensity-level differences between image subwindows centered at matching feature locations. A three-nearest-neighbor classifier using shape-context matching gave state-of-the-art classification accuracy on the MNIST database, with an error rate of only 0.63%.

The second dataset that we tried was the time-series dataset used in Vlachos et al. [2003]. To generate this dataset, various real datasets were used as seeds for generating a large number of time-series that are variations of the original sequences. Multiple copies of every real sequence were constructed by incorporating small variations in the original patterns, as well as additions of random compression and decompression in time. The final dataset contained a database set of 32,768 sequences, and a query set of 50 sequences. Sequences are multidimensional, with an average size of 500 points each. The series were normalized by subtracting the average value in each dimension. Exact distances were measured using constrained dynamic time warping, with a warping length $\delta = 10\%$ of the total length of the shortest sequence under comparison as described in Vlachos et al. [2003].

For the time-series dataset, we performed an initial evaluation on the 50 queries used as a test set in Vlachos et al. [2003]. Our method achieved a speedup factor of 51.2, with a 150-dimensional embedding and filter-and-refine parameter $p = 443$. With these settings, the true nearest-neighbor was retrieved correctly for each of the 50 queries. The indexing method in Vlachos et al. [2003] reports a speedup of approximately a factor of five while retrieving correctly the true nearest-neighbor for all 50 queries, measured on the same set of 50 queries we used.

However, to get a clearer picture of performance, we decided to use a larger set of queries. To achieve this, we merged the original query set with the database, and randomly splitted the merged set into a set of 1,000 query objects and a set of 31,818 database objects. Performance on the new set of queries was not as good as on the initial set of 50 queries; on the new query set, a speedup factor of

50 was obtained at the cost of missing the true nearest-neighbor for 10% of the queries. At the same time, even on the new set of queries, our method achieved significant speedups for k-nearest-neighbor retrieval with different accuracy percentages and different values of k. The results reported in the remainder of this section for the time-series dataset are with respect to the set of 1,000 queries.

7.3 Query-Sensitive versus Query-Insensitive

Since we use an adaptation of the BoostMap algorithm to construct query-sensitive embeddings, the most direct way to evaluate the advantages of query-sensitive embeddings is to compare them to the query-insensitive embeddings produced using the original BoostMap algorithm. In order to get a more comprehensive picture, we construct embeddings using both the optimization criterion proposed in Athitsos et al. [2004], which is based on the failure rate of the embedding on random training triples of objects, and the optimization criterion described in this article, which is based on the failure rate of the embedding on training triples constructed as discussed in Section 4.3.

Overall, then, we evaluate four embedding methods, each characterized by whether it is query-sensitive, and whether it uses random training triples. To denote each method and its relation to the other methods, we use the following abbreviations:

—*Ra* Training triples are chosen entirely randomly from the set of all possible triples, as in the original BoostMap method [Athitsos et al. 2004].

—*Se* Training triples are chosen selectively from a restricted set of possible triples, as described in Section 4.3.

—*QI* A query-insensitive distance measure D_{out} is constructed using the original BoostMap method.

—*QS* A query-sensitive distance measure D_{out} is constructed as proposed in this article.

Based on these abbreviations, Ra-QI denotes the original BoostMap algorithm, and Se-QS denotes the algorithm we describe in this article. Ra-QS and Se-QI add to the original BoostMap, respectively, the method for building a query-sensitive distance measure and theat for choosing training triples.

In Figures 5 and 6 we compare the four different methods on k-nearest-neighbor retrieval. The optimal number of exact distance computations (i.e., corresponding to optimal settings for the dimensionality of the embedding and parameter p) is shown for different values of k, from 1 to 50, and different percentages of accuracy (i.e., 90%, 95%, and 99%) in Figure 5 for the MNIST dataset and Figure 6 for the time-series dataset.

The results demonstrate that query-sensitive methods clearly outperform their query-insensitive counterparts, and provide significantly better trade-offs between efficiency and accuracy. In some cases, query-sensitive embeddings achieve performance that is two or three times as fast for a fixed error rate.

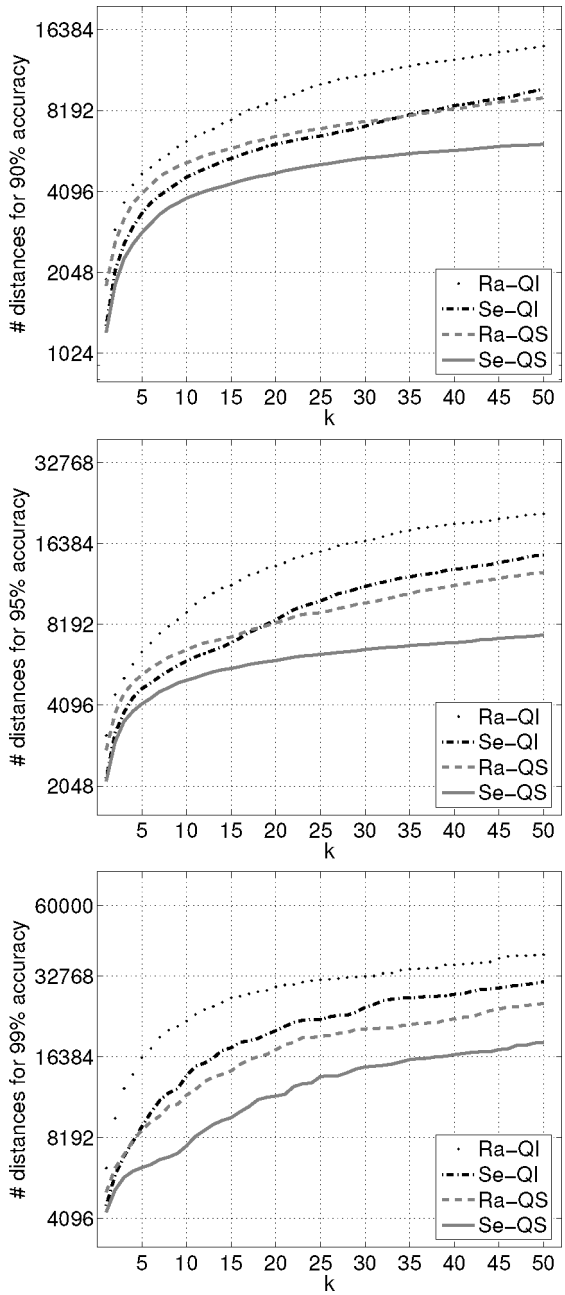

Fig. 5. Comparing methods Ra-QI, Ra-QS, Se-QI, and Se-QS (the proposed method) on the MNIST database, using shape-context matching as the exact distance measure. We show the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 10,000 query objects that we use as a test set.

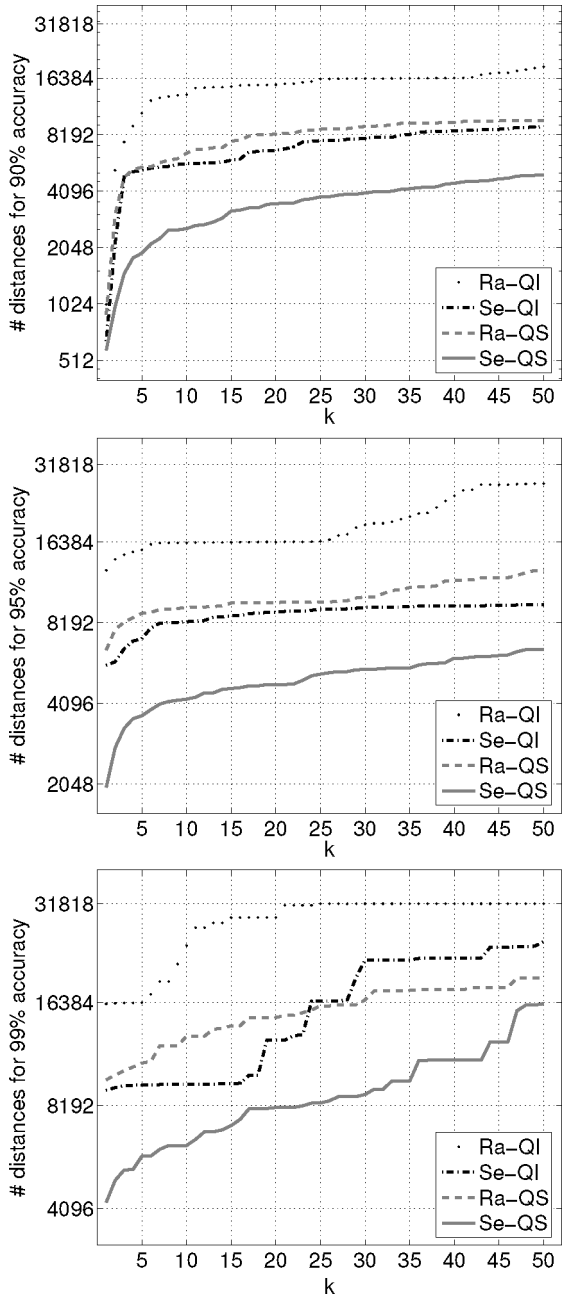

Fig. 6. Comparing methods Ra-QI, Ra-QS, Se-QI, and Se-QS (the proposed method) on the time-series database. We show the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 1,000 query objects that we use as a test set.

To train embeddings for the original BoostMap, the proposed method, and intermediate methods Ra-QS and Se-QI, we always used a training set of 10 million triples generated from a set X_{tr} of 5,000 database objects. At each training round, a sample of 30,000 training triples was used. Half of the sampled triples were triples misclassified by the current strong classifier. The set C of candidate objects also consisted of 5,000 database objects. Query objects from the test set were not used in any part of the training algorithm. Parameter k_1, used in choosing training triples, was set to five for the MNIST dataset and nine for the time series dataset, following the guidelines described in Section 4.3 for $k_{\max} = 50$. In this way, embeddings were optimized for the retrieval of up to 50 nearest neighbors per query.

7.4 Comparison to Alternative Indexing Methods

In order to better evaluate the performance of query-sensitive embeddings, we compare them to four alternative indexing methods:

—Fastmap [Faloutsos and Lin 1995]: In each dataset, we constructed a FastMap embedding by running the FastMap algorithm on a subset of the database containing 5,000 objects. This was the same set that we used as the set C of candidate objects in the training algorithm for our method.
—Random reference objects: We constructed a high-dimensional embedding for which each dimension was a 1D reference-object embedding, and the reference objects were chosen randomly.
—Random line projections: We constructed a high-dimensional embedding for which each dimension was a 1D line-projection embedding, and the pivot objects for this projection were chosen randomly.
—VP-trees [Yianilos 1993]: VP-trees rely on the triangle inequality to achieve efficient retrieval while always finding the true nearest-neighbors. Both shape-context matching and dynamic time warping do not obey the triangle inequality. Using a method similar to Sahinalp et al. [2003], we modify the search algorithm to guarantee correct results, assuming that the triangle inequality is satisfied up to a constant γ. In order to better evaluate the tradeoffs between efficiency and accuracy that VP-trees provide, we created a large number of different VP-trees using different values for γ. Larger values of γ lead to more accurate results and slower retrieval time.

In Figures 7 and 8 we compare the four different methods on k-nearest-neighbor retrieval. The optimal number of exact distance computations (i.e., corresponding to optimal settings for the dimensionality of the embedding and the parameter p) is shown for different values of k, from 1 to 50, and different percentages of accuracy (i.e., 90%, 95%, and 99%). For VP-trees, for each specified accuracy, we found by trial-and-error the smallest γ value that provided this accuracy. In many settings, our method is faster than any alternative by a factor between 1.45 and 2.7, for example, for 90%, 95%, and 99% accuracy on 1, 10, and 50-nearest-neighbor retrieval on the MNIST database.

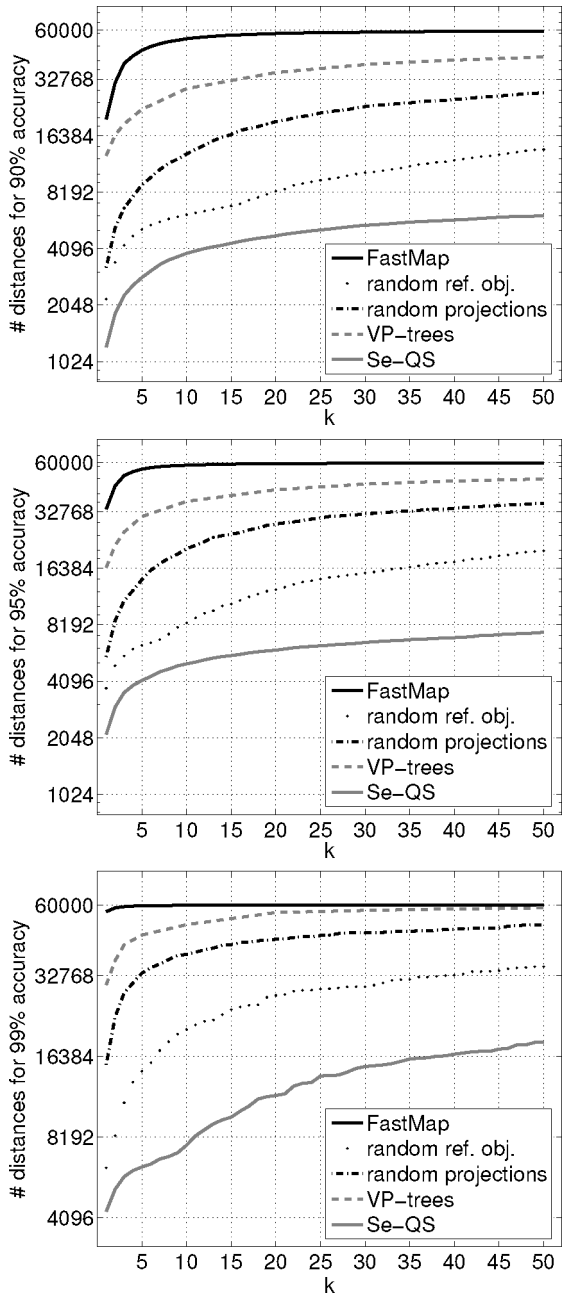

Fig. 7. Comparing methods Se-QS (our method), FastMap, random reference objects, random line-projections, and VP-trees on the MNIST database. We show the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 10,000 query objects that we use as a test set.

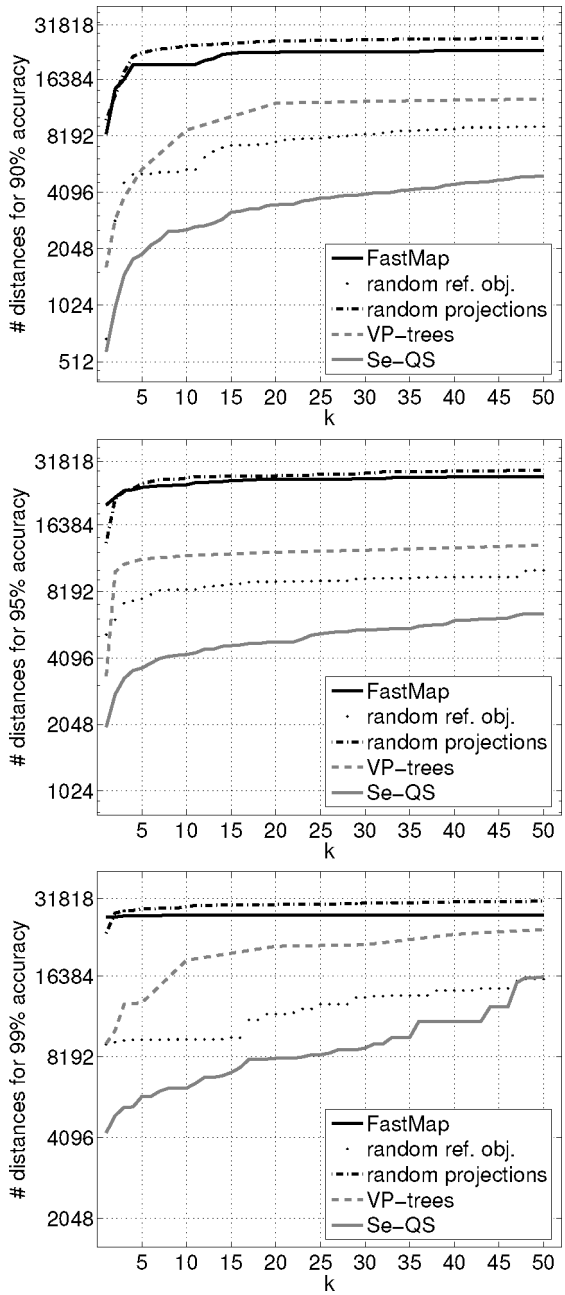

Fig. 8. Comparing methods Se-QS (our method), FastMap, random reference objects, random line-projections, and VP-trees, on the time-series database. We show the number of exact distance computations needed by each method to achieve correct retrieval of all k nearest neighbors (k ranging from 1 to 50) for 90%, 95%, and 99% of the 1,000 query objects that we use as a test set.

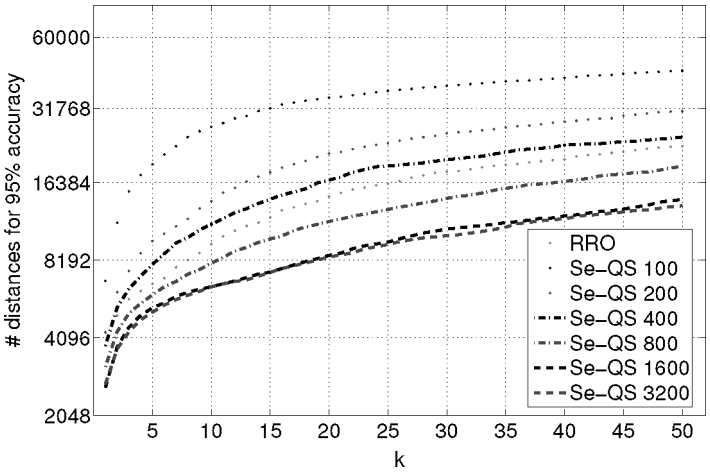

Fig. 9. Retrieval results on the MNIST dataset for 100-dimensional Se-QS embeddings constructed with sets C and X_{tr}, including only 100, 200, 400, 800, 1,600, and 3,200 objects, respectively, and using only 10,000 training triples. We compare these results to those for a 100-dimensional RRO embedding. For different values of k, the figure shows the number of exact distance computations required by each embedding method in order to retrieve the true k nearest neighbors for 95% of the 10,000 queries.

7.5 Experiments on Training Using Small Database Samples

We have run an experiment on the MNIST dataset, in which we used the proposed method, namely, method Se-QS, but with relatively small sizes for sets C (used to define 1D embeddings) and X_{tr} (used to form training triples) used in the training algorithm, and with fewer training triples. Set sizes $|C|$ and $|X_{\text{tr}}|$ were always equal in this experiment, and we only used 10,000 training triples. We constructed 100-dimensional query-sensitive embeddings by setting $|C|$ and $|X_{\text{tr}}|$ to each of the following values: 100, 200, 400, 800, 1,600, and 3,200. In Figure 9, we compare the results of these embeddings to each other and to the results of a 100-dimensional RRO embedding. We chose the RRO method for comparison because it gave the best results among all non-BoostMap-based methods on the MNIST dataset.

Let n be the number of database objects, and define quantity $M(F)$ for an embedding F to be equal to the number of exact distance computations needed to embed a single object. For a d-dimensional RRO embedding F, $M(F) = d$. For a query-sensitive d-dimensional embedding F constructed using d_1 reference-object embeddings and d_2 line-projection embeddings, $M(F) = d_1 + 2d_2$. Since $|C| = |X_{\text{tr}}|$, the number of exact distances we need to compute for constructing a query-sensitive embedding F is $2|C|^2 + nM(F)$. We need $2|C|^2$ distance computations to compute all the distances necessary to run the training algorithm, and $nM(F)$ distances to embed all database objects once we have constructed the embedding. In contrast, for an RRO d-dimensional embedding F, we only need $nM(F) = nd$ distances, since no training is performed. For the 100-dimensional RRO embedding, we need to compute 6 million distances, which takes 111 hours on an AMD Opteron 2.2GHz processor.

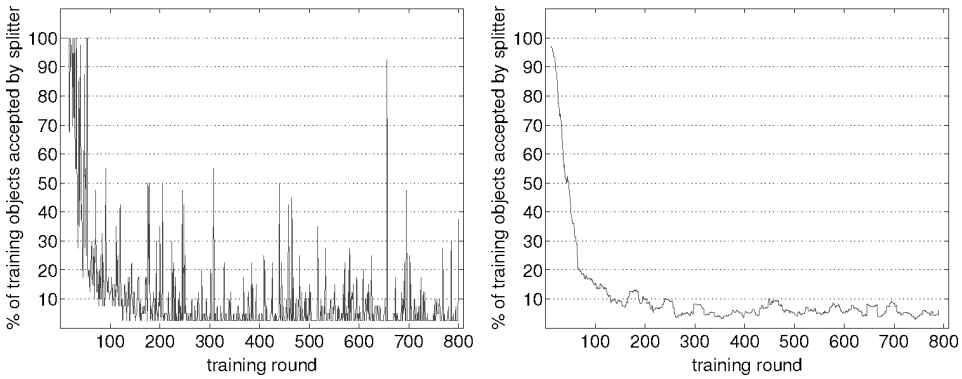

Fig. 10. (left): Plotting the percentage of database objects accepted by the splitter vs. the training round at which the splitter was chosen, for the MNIST dataset. (right): a smoothed version of the top figure. Here, we show for each training round the average percentage of objects accepted by the splitters chosen at this round, the ten previous rounds, and ten subsequent rounds.

For the 100-dimensional query-sensitive embeddings constructed in this experiment, $M(F)$ ranged between 130 and 165. The first value of $|C|$ for which a Se-QS embedding outperforms the RRO embedding is $|C| = 800$. For this embedding, $M(F) = 162$, and to construct this embedding, in total we needed to compute exactly 11 million distances. On our computer it takes about 204 hours to compute that many distances, and it takes about 20 additional minutes to perform the training. Therefore, for less than double the processing time of constructing an RRO embedding, we can construct a query-sensitive embedding that performs better. As we see in Figure 9, increasing $|C|$ to 1,600 leads to even better results, after which, however, increasing $|C|$ further does not improve performance significantly.

It is also important to note here that setting $|C|$ to 100, 200, or 400, the training algorithm creates embeddings which perform worse than choosing random reference-objects. These results demonstrate the problem of overfitting: When the training set is too small, the training algorithm can fit the training data very well, but performance on unseen data is much worse.

7.6 Measuring the Range of Query-Sensitive Classifiers

Here we take a closer look at the training algorithm as applied on the MNIST dataset. We denote by $\tilde{Q}_{F'_j,V_j}$ the weak classifier chosen by the training algorithm at training round j. For each j, we measured the percentage of database objects accepted by splitter $S_{F'_j,V_j}$. This percentage is an indicator of how *local* the classifier $\tilde{Q}_{F'_j,V_j}$ is, that is, for what fraction of triples (q, a, b) classifier $\tilde{Q}_{F'_j,V_j}$ produces a nonzero output and thus contributes to the output of the strong classifier H constructed by AdaBoost. The results are shown on Figure 10. As can be seen in the figure, the first 20–30 classifiers chosen were global or almost global, and gradually became more and more local. For the last 500 training rounds, each weak classifier chosen at these rounds was applicable, on average, to fewer than 10% of all possible triples, meaning that the corresponding splitter accepted fewer than 10% of all database objects.

Table I. Training Time vs. Classification Accuracy of the Classifier

MNIST Database with Shape Context		
	sampling	no sampling
error rate on training triples	1.49 %	0.00 %
error rate on validation triples	1.48 %	1.51 %
training time	64 minutes	633 minutes

Here, we compare the training time (to construct a 300-dimensional embedding) and classification accuracy of the resulting classifier on training and validation triples, for two variations of the training algorithm: the "sampling" variation which uses 10 million training triples and samples 30,000 of them at each training round, and the "no sampling" variation, where the same 300,000 training triples are used at each round. Results are shown for the MNIST dataset.

What this picture tells us is that on the MNIST dataset, for the first 20 or 30 dimensions, using a query-sensitive distance measure does not make much difference because for almost all queries, all the dimensions turn out to be selected. From the point of view of a particular query object q, the first 20 or 30 embedding dimensions all provide useful information about the nearest neighbors of this query. However, after a few tens of training rounds, given the information provided by the weak classifiers which have already been chosen, it becomes less and less likely that the next chosen weak classifier will provide beneficial information about the nearest neighbors of q. According to the training algorithm, it is usually better to ignore this information, and instead only use the information from those weak classifiers that were considered relevant for this query. Embedding methods, like the original BoostMap algorithm, which use query-insensitive distance measures do not have the option of ignoring this information, and the filter-and-refine experiments indicate that such insensitivity leads to a notable decrease in accuracy.

7.7 Runtimes

On average, computing exact shape-context distances can be done at the rate of 15 distances per second, and computing constrained dynamic time warping distances can be done at the rate of about 60 distances per second, both on an AMD Opteron 2.2GHz processor using an optimized C++ implementation. To obtain the corresponding processing times per query for each setting shown in Figures 5, 6, 7, 8, and 9, we simply need to divide the number of exact distance computations by 15 for shape-context matching and by 60 for dynamic time warping.

With respect to the training algorithm, it typically takes about one hour to construct a 300-dimensional embedding, using 10 million training triples and a sample of 30,000 of these triples at each training round. The training algorithm is about ten times faster than the implementation we used in Athitsos et al. [2005], where we used 300,000 training triples and no sampling, while the accuracy of the classifiers obtained with the current implementation is marginally better, as measured on the set of validation triples. As an example, Table I compares training times and classification accuracies obtained with and without sampling for the MNIST dataset. Without sampling, the classifier achieves a zero error rate on training triples, but the error rate on validation triples is slightly higher than using sampling.

Table II. Stress Values for Query-Insensitive and Query-Sensitive Embeddings

Dataset	Query-insensitive stress	Query-sensitive stress
MNIST	0.1447	.1364
Time series	0.3063	.1773

These stress values were obtained on the MNIST and time-series datasets.

7.8 Experiments on Minimizing Stress

We have applied to our two datasets the algorithms described in Sections 6.1 and 6.2 for constructing, respectively, query-insensitive and sensitive embeddings so as to minimize embedding stress. For both algorithms, the pool of 1D embeddings consisted of 5,000 reference-object embeddings, defined using 5,000 database objects. During embedding construction, 100,000 training pairs of objects were used to measure stress. The objects in the training pairs were from a subset of 5,000 database objects, disjoint from the database objects used as reference objects. For each 1D embedding F, the set $\mathbb{V}_F$ of candidate areas of influence had size 50. The stopping criterion used in the embedding construction algorithms was that j could not exceed 500, that is, 500 reference objects could be picked. Once an embedding was constructed, its stress was measured based on the distances from all test to all database objects.

Table II lists the results obtained using insensitive and query-sensitive embeddings on the MNIST and time-series datasets. In both datasets, query-sensitive embeddings achieved a smaller stress value. In the MNIST dataset, the query-sensitive stress was 5.7% smaller than the query-insensitive stress. The difference was significantly more pronounced in the time-series dataset, where query-sensitive stress was about 42% smaller than query-insensitive stress. As in the nearest-neighbor retrieval experiments, we see that query-sensitive embeddings have provided better results than their query-insensitive counterparts.

8. DISCUSSION AND CONCLUSIONS

We have presented a novel type of embedding for nearest-neighbor retrieval. In particular, we have described how to construct query-sensitive embeddings, namely, embeddings that use a query-sensitive distance measure for the target space of the embedding. Such a distance measure can capture the fact that different embedding dimensions are important for different queries, and thus leads to improved retrieval accuracy at the filter step of filter-and-refine retrieval. Overall, using a query-sensitive distance measure increases the modeling power of the embedding, and allows better approximations of the original space.

The experimental results reported in this article provide a quantitative comparison of the proposed algorithm to several alternative methods, on two datasets: the MNIST dataset of handwritten digits using shape-context matching as the underlying distance measure, and a time-series dataset using constrained dynamic time warping as the underlying distance measure. The experiments demonstrate that the proposed method clearly outperforms the

alternative methods we have tested, and that adding query-sensitivity clearly improves the quality of the embedding.

We are interested in expanding our theoretical understanding of query-sensitive embeddings and their properties. The fact that any finite metric space has an isometric query-sensitive embedding demonstrates the additional modeling power of query-sensitive embeddings, although this fact cannot be directly applied to provide any guarantees of performance in a filter-and-refine retrieval system. It is an open question whether we can leverage the flexibility of query-sensitive embeddings to provide guarantees of embedding quality and retrieval performance that query-insensitive embeddings cannot match.

We believe that query-sensitive distance measures may prove useful in other settings, in addition to embedding-based nearest-neighbor retrieval. A common problem in data mining, clustering, and pattern recognition applications is how to construct a meaningful distance measure for comparing high-dimensional vectors. We are interested in exploring whether our algorithm for learning a query-sensitive distance measure can offer advantages in such applications.

ACKNOWLEDGMENTS

This work was supported by NSF grants IIS-0308213 and IIS-0133825, and by ONR grant N00014-03-1-0108.

REFERENCES

AGGARWAL, C. C. 2001. Re-Designing distance functions and distance-based applications for high dimensional data. *SIGMOD Rec. 30*, 1, 13–18.

ATHITSOS, V. 2006. Learning embeddings for indexing, retrieval, and classification, with applications to object and shape recognition in image databases. Ph.D. thesis, Boston University.

ATHITSOS, V., ALON, J., AND SCLAROFF, S. 2005. Efficient nearest neighbor classification using a cascade of approximate similarity measures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 486–493.

ATHITSOS, V., ALON, J., SCLAROFF, S., AND KOLLIOS, G. 2004. BoostMap: A method for efficient approximate similarity rankings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 268–275.

ATHITSOS, V., HADJIELEFTHERIOU, M., KOLLIOS, G., AND SCLAROFF, S. 2005. Query-Sensitive embeddings. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 706–717.

ATHITSOS, V. AND SCLAROFF, S. 2003. Database indexing methods for 3D hand pose estimation. In *Proceedings of the Gesture Workshop*. (Heidelberg, Germany) Springer Verlag. 288–299.

BARROW, H., TENENBAUM, J., BOLLES, R., AND WOLF, H. 1977. Parametric correspondence and Chamfer matching: Two new techniques for image matching. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 659–663.

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2002. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 4, 509–522.

BÖHM, C., BERCHTOLD, S., AND KEIM, D. A. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv. 33*, 3, 322–373.

BOURGAIN, J. 1985. On Lipschitz embeddings of finite metric spaces in Hilbert space. *Israel J. Math. 52*, 46–52.

BOZKAYA, T. AND ÖZSOYOGLU, Z. 1999. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst. 24*, 3, 361–404.

CHAKRABARTI, K. AND MEHROTRA, S. 2000. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proceedings of the International Conference on Very Large Data Bases*. 89–100.

DOMENICONI, C., PENG, J., AND GUNOPULOS, D. 2002. Locally adaptive metric nearest-neighbor classification. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 9, 1281–1285.

EGECIOGLU, Ö. AND FERHATOSMANOGLU, H. 2000. Dimensionality reduction and similarity distance computation by inner product approximations. In *Proceedings of the International Conference on Information and Knowledge Management*. 219–226.

FALOUTSOS, C. AND LIN, K. I. 1995. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 163–174.

GIONIS, A., INDYK, P., AND MOTWANI, R. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Databases*. 518–529.

HASTIE, T. AND TIBSHIRANI, R. 1996. Discriminant adaptive nearest-neighbor classification. *IEEE Trans. Pattern Analy. Mach. Intell. 18*, 6, 607–616.

HINNEBURG, A., AGGARWAL, C. C., AND KEIM, D. A. 2000. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the International Conference on Very Large Data Bases*. 506–515.

HJALTASON, G. AND SAMET, H. 2003a. Properties of embedding methods for similarity searching in metric spaces. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 5, 530–549.

HJALTASON, G. R. AND SAMET, H. 2003b. Index-Driven similarity search in metric spaces. *ACM Trans. Database Syst. 28*, 4, 517–580.

HRISTESCU, G. AND FARACH-COLTON, M. 1999. Cluster-Preserving embedding of proteins. Tech. Rep. 99-50, Department of Computer Science, Rutgers University.

HUTTENLOCHER, D., KLANDERMAN, D., AND RUCKLIGE, A. 1993. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell. 15*, 9, 850–863.

JACOBS, D. W., WEINSHALL, D., AND GDALYAHU, Y. 2000. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 6, 583–600.

KANTH, K. V. R., AGRAWAL, D., AND SINGH, A. 1998. Dimensionality reduction for similarity searching in dynamic databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 166–176.

KEOGH, E. 2002. Exact indexing of dynamic time warping. In *Proceedings of the International Conference on Very Large Data Bases*. 406–417.

KOUDAS, N., OOI, B. C., SHEN, H. T., AND TUNG, A. K. H. 2004. LDC: Enabling search by partial distance in a hyper-dimensional space. In *Proceedings of the IEEE International Conference on Data Engineering*. 6–17.

KRUSKALL, J. B. AND LIBERMAN, M. 1983. The symmetric time warping algorithm: From continuous to discrete. In *Time Warps*. Addison-Wesley.

LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. 1998. Gradient-Based learning applied to document recognition. *Proc. IEEE 86*, 11, 2278–2324.

LI, C., CHANG, E., GARCIA-MOLINA, H., AND WIEDERHOLD, G. 2002. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Trans. Knowl. Data Eng. 14*, 4, 792–808.

PAREDES, R. AND VIDAL, E. 2000. A class-dependent weighted dissimilarity measure for nearest neighbor classification problems. *Pattern Recogn. Lett. 21*, 12, 1027–1036.

ROWEIS, S. AND SAUL, L. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Sci. 290*, 2323–2326.

SAHINALP, S. C., TASAN, M., MACKER, J., AND ÖZSOYOGLU, Z. M. 2003. Distance based indexing for string proximity search. In *Proceedings of the IEEE International Conference on Data Engineering*. 125–136.

SAKURAI, Y., YOSHIKAWA, M., UEMURA, S., AND KOJIMA, H. 2000. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of the International Conference on Very Large Data Bases*. 516–526.

SCHAPIRE, R. AND SINGER, Y. 1999. Improved boosting algorithms using confidence-rated predictions. *Mach. Learn. 37*, 3, 297–336.

TENENBAUM, J., SILVA, V. D., AND LANGFORD, J. 2000. A global geometric framework for nonlinear dimensionality reduction. *Sci. 290*, 2319–2323.

TRAINA, JR., C., TRAINA, A., SEEGER, B., AND FALOUTSOS, C. 2000. Slim-Trees: High performance metric trees minimizing overlap between nodes. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*. 51–65.

VLACHOS, M., HADJIELEFTHERIOU, M., GUNOPULOS, D., AND KEOGH, E. 2003. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 216–225.

WANG, X., WANG, J. T. L., LIN, K. I., SHASHA, D., SHAPIRO, B. A., AND ZHANG, K. 2000. An index structure for data mining and clustering. *Know. Inf. Syst. 2*, 2, 161–184.

WEBER, R. AND BÖHM, K. 2000. Trading quality for time with nearest-neighbor search. In *Proceedings of the International Conference on Extending Database Technology*. 21–35.

WEBER, R., SCHEK, H.-J., AND BLOTI, S. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large Data Bases*. 194–205.

WHITE, D. A. AND JAIN, R. 1996. Similarity indexing: Algorithms and performance. In *Proceedings of the SPIE Storage and Retrieval for Image and Video Databases*. 62–73.

YI, B.-K., JAGADISH, H. V., AND FALOUTSOS, C. 1998. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the IEEE International Conference on Data Engineering*. 201–208.

YIANILOS, P. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 311–321.

YOUNG, F. AND HAMER, R. 1987. *Multidimensional Scaling: History, Theory and Applications*. Lawrence Erlbaum, Hillsdale, NJ.

ZEZULA, P., SAVINO, P., AMATO, G., AND RABITTI, F. 1998. Approximate similarity retrieval with M-trees. *VLDB J. 4*, 275–293.

Received October 2005; revised May 2006; accepted October 2006