Properties of Embedding Methods for Similarity Searching in Metric Spaces

Gísli R. Hjaltason, *Member*, *IEEE Computer Society*, and Hanan Samet, *Fellow*, *IEEE*

Abstract—Complex data types—such as images, documents, DNA sequences, etc.—are becoming increasingly important in modern database applications. A typical query in many of these applications seeks to find objects that are similar to some target object, where (dis)similarity is defined by some distance function. Often, the cost of evaluating the distance between two objects is very high. Thus, the number of distance evaluations should be kept at a minimum, while (ideally) maintaining the quality of the result. One way to approach this goal is to *embed* the data objects in a vector space so that the distances of the embedded objects approximates the actual distances. Thus, queries can be performed (for the most part) on the embedded objects. In this paper, we are especially interested in examining the issue of whether or not the embedding methods will ensure that no relevant objects are left out (i.e., there are no false dismissals and, hence, the correct result is reported). Particular attention is paid to the SparseMap, FastMap, and MetricMap embedding methods. SparseMap is a variant of Lipschitz embeddings, while FastMap and MetricMap are inspired by dimension reduction methods for Euclidean spaces (using KLT or the related PCA and SVD). We show that, in general, none of these embedding methods guarantee that queries on the embedded objects have no false dismissals, while also demonstrating the limited cases in which the guarantee does hold. Moreover, we describe a variant of SparseMap that allows queries with no false dismissals. In addition, we show that with FastMap and MetricMap, the distances of the embedded objects can be much greater than the actual distances. This makes it impossible (or at least impractical) to modify FastMap and MetricMap to guarantee no false dismissals.

Index Terms—Embedding methods, metric spaces, similarity search, multimedia databases, contractiveness, distortion, quality, Lipschitz embeddings, singular value decomposition (SVD), SparseMap, FastMap, MetricMap.

1 Introduction

The need for being able to handle novel data types is becoming ever more important in modern database applications, including in such fields as bioinformatics, computer vision, pattern recognition, image databases, computer-aided design (CAD), etc. Examples of this type of data include images, video, and text documents, and even such exotic data as protein and DNA sequences. A common type of query on such data, known as similarity search (also termed *content-based* or *similarity* retrieval), seeks to find objects in the database that are similar to some target object. For such queries to be meaningful, some measure of similarity between the objects in the database must be defined. Usually, the query returns objects having at least some given level of similarity to the target object (range query), or some given number of the most similar objects (nearest-neighbor query).

The rest of this section is organized as follows: In Section 1.1, we describe how mapping a set of data objects into a vector space can make search more efficient. In Section 1.2, we present in more detail the general class of embeddings that we focus on, namely, ones that do not depend on knowledge of the specific nature of the data objects. In Section 1.3, we discuss similarity search using embedded data objects. Finally, in Section 1.4, we present an outline of the remainder of this paper.

1.1 Embedding

The level of similarity (or, actually, dissimilarity) between two objects is typically measured with a distance function d. The challenge faced in performing similarity search is that the distance function is often very expensive to compute for complex data objects. For example, computing the similarity of two proteins, based on their amino acid sequences, has been reported as taking several hundred milliseconds on a typical workstation [1]. For this reason, it is desirable to make as few distance calculations as possible when executing similarity queries, preferably none. A common approach to achieving this goal is to map, or *embed*, the set of objects into points in a low-dimensional *embedding space* and then conduct the search there with the help of multidimensional indexing methods [2], [3]. Intuitively, the rationale for performing such mappings is that distances in the embedding space approximate distances of the objects in the original space and that searching in the embedding space is "less expensive" than in the original space.

In this paper, we focus on embedding methods that produce a mapping solely based on the distance function d, which is usually required to be a distance metric (i.e., d must satisfy the triangle inequality, or, at worst, only a relatively small number of the distances violate the triangle inequality). The advantage of such methods is that they can be applied without knowledge of the specific nature of the data objects since the distance function d is treated as a "black box." Nevertheless, domain-specific mapping methods exist for a variety of data types and applications, such as for images [4], [5], [6] and time series [7], [8]. Such methods are often referred

- *G.R. Hjaltason is with the School Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada. E-mail: gisli@db.uwaterloo.ca.*
- *H. Samet is with the Computer Science Department, Center for Automation Research, and Institute for Advanced Computer Studies at the University of Maryland, College Park, Maryland 20742. E-mail: hjs@cs.umd.edu.*

Manuscript received 16 Oct. 2001; revised 26 July 2002; accepted 18 Sept. 2002. Recommended for acceptance by P. Meer.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 115209.

to as *feature extraction* and their results as *feature vectors*.[1] For some of these methods, no explicit distance function d may exist beyond the subjective judgments of human domain experts. Naturally, when we have both an explicit distance function and a domain-specific mapping method, we have a choice as to how to embed the objects.

An important class of embedding methods is one where the data objects are vectors of high dimensionality, to which we wish to apply *dimensionality reduction*. Examples of such methods include Karhunen-Loève transform (KLT) [9], also known as principal component analysis (PCA) [10], discrete Fourier transform (DFT) [11], wavelet transforms [12], [13], and random projections [14], [15] (also, see [16] for a survey of some of these methods, as well as of some domain-specific mapping methods). Many of these methods require d to be the Euclidean metric, but some also work for other metrics (e.g., weighted Euclidean metric). In contrast, the methods we focus on below can be applied for arbitrary distance metrics (but, see Section 1.4), and may even be competitive alternatives when pure dimension reduction methods are applicable.

1.2 General Distance Metrics

A tuple (S, d) is said to be a *finite metric space* if S is a finite set of cardinality N and $d : S \times S \rightarrow \mathbb{R}^+$ is a distance metric. A great deal of work has been done on embedding finite metric spaces into low-dimensional real-normed spaces—that is, real-valued vector spaces with a norm, which serves as the basis of a distance metric. Such embeddings have been extensively studied in pure mathematics [17], [18] (where the embedding is often performed into Hilbert spaces, which are essentially abstractions of real-valued vector spaces with a dot product [19]), and have found application in a variety of settings [1], [20], [21], [22], [23]. Usually, the norm is one of the L_p norms, $||x||_p = (\sum |x_i|^p)^{1/p}$. Distance metrics based on such a norm are often termed *Minkowski metrics* when $p \geq 1$. The most common Minkowski metrics are the Euclidean distance metric (L_2), the City Block distance metric (L_1), and the Chessboard distance metric (L_∞), denoted below by d_E, d_A, and d_M (for Euclidean, average, and maximum), respectively.

Formally, an embedding of a finite metric space (S, d) into $(\mathbb{R}^k, \delta)$ is a mapping $F : S \rightarrow \mathbb{R}^k$, where k is the dimensionality of the embedding space and $\delta : \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}^+$ is the distance metric of the embedding space. If we denote the norm in $\mathbb{R}^k$ by $||\cdot||$, the distance metric δ is defined as $\delta(x, y) = ||x - y||$. Ideally, the distance $\delta(F(o_1), F(o_2))$ in the embedding space adheres closely to the distance $d(o_1, o_2)$ in the original space. However, it is often impossible and/or impractical to achieve exact correspondence between the distances based on d and δ. If an embedding F exists such that $\delta(F(o_1), F(o_2)) = d(o_1, o_2)$ for all $o_1, o_2 \in S$, then (S, d) and $(\mathbb{R}^k, \delta)$ are said to be *isometric* (strictly speaking, (S, d) is isometric to $(F(S), \delta)$, where $F(S) \subset \mathbb{R}^k$ is the image of S under F).

In similarity retrieval applications, it is typical to consider a larger, potentially infinite, metric space $(\mathbb{U}, d)$, where $S \subset \mathbb{U}$, and to define the domain of F as $\mathbb{U}$ rather than S. As a concrete example, $\mathbb{U}$ might be the set of all possible protein sequences, while $S \subset \mathbb{U}$ is a particular data set under study.

1. The terms vectors and points are often used interchangeably; formally, a point p gives rise to a vector $\mathbf{v} = \vec{0p}$ with the same coordinate values, where 0 is the origin.

The reason for generalizing F is that the query object $q \in \mathbb{U}$ used for similarity queries is typically not a member of S. Furthermore, in many applications, the database is dynamic, so that objects may be inserted into and removed from S over time. Note that, even though F can be applied on the entire set $\mathbb{U}$, the embedding F is still usually constructed with the goal of approximating the distances of only the objects in S. Thus, F may not provide good distance preservation for objects in $\mathbb{U} \setminus S$, although we would hope that it performs adequately. Nevertheless, if many objects are added to the data set S, the quality of the embedding will tend to degenerate. Unfortunately, recomputing F may cause the value of $F(o)$ to change for every existing object $o \in S$, requiring a costly rebuilding of any index that has been built on the embedding space. Thus, it is preferable to recompute F only infrequently rather than for each update of S, e.g., based on the number of updates or when the "quality" of F falls below some threshold. An example of a quality measure is distortion [1] which measures how much larger or smaller the distances in the embedding space are than the corresponding distances in the original space (see Section 2.2 for more details).

1.3 Similarity Searching

Unless the distances measured with the distance function δ in the embedding space correspond exactly to the distances measured with the original distance function d, queries performed in the embedding space clearly do not have the same accuracy as queries performed in the original metric space. In particular, if R_o is the set of objects resulting from a similarity query performed in the original metric space, and R_e is the set of objects resulting from the corresponding query in the embedding space, then some of the objects in R_o may not be present in R_e, and vice versa. In other words, some objects that should be in the result R_o are not found in R_e, while other objects that should not be in the result R_o are found in R_e. The assumption, of course, is that R_o is the correct result. The notion of *precision* captures the proportion of objects in R_e that are in the correct result R_o, and is defined as $\frac{|R_e \cap R_o|}{|R_e|}$. The notion of *recall*, on the other hand, captures the proportion of the correct result R_o found in R_e, and is defined as $\frac{|R_e \cap R_o|}{|R_o|}$. When the precision is 100 percent, all the objects in R_e are correct. However, note that a precision of 100 percent does not necessarily mean that R_e contains every element of the correct result. On the other hand, when the recall is 100 percent, all correct objects occur in R_e (as well as possibly some that are not correct!).

There are two options in making use of R_e, the result from a query in the embedding space. One option is to use it "as is" and report R_e to the user as the result of the query. Unfortunately, the precision and recall of queries in the embedding space are often unknown, except perhaps experimentally. The other option is to use a *filter and refine* strategy, where the query in the embedding space, resulting in R_e, is used as a "filter" and the actual distances, as measured by d, are used to "refine" the result, thereby forming the set R_f. In practice, the filter and refine steps are often interleaved (see Section 2.3). A filter and refine strategy allows the removal of all irrelevant objects from the result and, thus, enables bringing the precision up to 100 percent. However, if there are objects in R_o that are missing in R_e, then they will clearly also be missing in R_f. Thus, unless the recall of queries in the embedding space is 100 percent, it is

impossible to achieve 100 percent recall even with refinement (unless we resort to the expensive strategy of restarting the query in the embedding space with less selective criteria).

As will be shown in Section 2.3, the property that an embedding is *contractive* is sufficient to guarantee 100 percent recall of queries in the embedding space. Briefly, this property implies that distances in the embedding space lower-bound distances in the original space; see Section 2.3.

1.4 Outline

In this paper, we review three general embedding methods, SparseMap, FastMap, and MetricMap, for each one presenting a detailed description aimed at conveying the intuition and motivation behind each method. For each method, we also examine whether they allow achieving 100 percent recall and, in some cases, suggest variations of the methods aimed at improving some of their characteristics. Some further details can be found in [24], such as an application of the methods on a sample data set and proofs of many of the stated properties.

The rest of the paper is organized as follows: In Section 2, we discuss the properties of an embedding F and methods for measuring the quality of F. We find that 100 percent recall is assured when F is *contractive*, which implies that the distances in the embedding space lower-bound the corresponding distances in the original space; see Section 2.3. In Section 3, we describe SparseMap and the Lipschitz embeddings that it is based on and introduce modifications that make the result of SparseMap contractive. In Section 4, we review FastMap, while in Section 5, we show that FastMap does not, in general, yield contractive embeddings (and cannot be modified to do so in a practical manner). In Section 6, we discuss MetricMap and show its relationship to singular value decomposition (SVD). In Section 7, we show that MetricMap also does not guarantee a contractive embedding, propose some modifications to the method, and explore the relationship between MetricMap and FastMap. Finally, in Section 8, we draw some conclusions and present directions for future research.

2 Properties of Embeddings

In this section, we first describe basic properties of embeddings and why embeddings are useful (Section 2.1). Next, we outline a number of different ways of measuring the quality of embeddings (Section 2.2). We conclude with a discussion of important properties of embeddings that affect similarity queries and sketch how these properties can be exploited for both range queries and nearest-neighbor queries (Section 2.3).

2.1 Basic Properties

As mentioned in Section 1, embedding complex data objects into low-dimensional vector spaces facilitates similarity queries in an environment where we are given a set S of N objects and a function d indicating the distances between them. At times, this distance function is represented by an $N \times N$ similarity matrix containing the distance between all $N(N-1)/2$ distinct pairs of objects. The justification for applying embeddings is that for any finite metric space (S, d), we can usually find a function F that maps the N objects into a vector space of dimensionality k, given a sufficiently high value of k, such that the distances between the points are approximately preserved when using a distance function δ in the k-dimensional space. In other words, for any pair of objects a and b, we have $d(a,b) \approx \delta(F(a), F(b))$. In practice, our goal is to use a relatively low value for k in the mapping (i.e., $k \ll N$), thereby allowing effective use of multidimensional indexing methods, while still achieving reasonable distance preservation. Since distance computation can be expensive, the mapping F should ideally be computed efficiently (i.e., require substantially fewer than $O(N^2)$ distance computations), should preserve distances to a reasonable extent, and should provide a fast way of obtaining the k-dimensional point corresponding to a query object (usually an object not in S).

2.2 Measuring Quality

A number of different ways have been proposed for measuring the quality of an embedding procedure (i.e., a method that constructs a mapping F) or of a particular embedding F produced by such a procedure. The concept of *distortion* (e.g., [1]) is frequently used for this purpose. Distortion measures how much larger or smaller the distances in the embedding space $\delta(F(o_1), F(o_2))$ are than the corresponding distances $d(o_1, o_2)$ in the original space. In particular, the distortion is defined as the lowest value $c_1 c_2$ that guarantees that

$$\frac{1}{c_1} \cdot d(o_1, o_2) \leq \delta(F(o_1), F(o_2)) \leq c_2 \cdot d(o_1, o_2), \qquad (1)$$

for all pairs of objects $o_1, o_2 \in S$, where $c_1, c_2 \geq 1$. In other words, the distance values $\delta(F(o_1), F(o_2))$ in the embedding space may be as much as a factor of c_1 smaller and a factor of c_2 larger than the actual distances $d(o_1, o_2)$. Note that, for some embedding procedures, there may be no upper or lower bound on the distance ratio for the embeddings that they construct, so c_1 and/or c_2 may be infinite in this case. Of course, the distortion is always bounded when considering any given embedding F and finite metric space (S, d). A number of general results are known about embeddings, e.g., that any finite metric space can be embedded in Euclidean space with $O(\log N)$ distortion [1].

Another common measure of a particular embedding F with respect to a data set S is *stress* [22]. Stress measures the overall deviation in the distances (i.e., the extent to which they differ), and is typically defined in terms of variance:

$$\frac{\sum_{o_1,o_2} \left(\delta(F(o_1), F(o_2)) - d(o_1, o_2)\right)^2}{\sum_{o_1,o_2} d(o_1, o_2)^2}.$$

Alternative definitions of stress may be more appropriate for certain applications. For example, the sum in the denominator may be on $\delta(F(o_1), F(o_2))^2$, or the division by $d(o_1, o_2)^2$ may occur inside the sum, instead of in a separate sum.

A measure of the quality of embeddings that has been proposed in clustering applications [21] is termed Cluster Preservation Ratio (CPR). This measure can be applied to a data set when a known clustering exists for the objects. In this case, CPR indicates the average ratio of cluster preservation over all objects in the data set. In other words, for each object o whose cluster is of size s, we find the s nearest neighbors in the embedding space and compute the fraction of cluster objects that are among these s neighbors. For an example where the clustering of the objects is known, consider the situation of proteins in a computational biology application [21]. In particular, a number of proteins have been studied extensively in terms of their biochemical functions, so proteins having similar functions can be grouped together. Therefore,

we can test whether amino acid sequences representing these known proteins follow this grouping.

Finally, we want to point out that precision and recall (as defined in Section 1.3) of a similarity query performed in the embedding space can also be used as measures of the quality of embeddings. Ideally, both measures should be close to 100 percent, but poor distance preservation will decrease both. Notice that precision and recall differ from the other measures in that the query object q is not in S and, thus, q was not taken into account when the embedding F was constructed (although F can be applied to q once F has been constructed). Thus, we may get drastically different precision and recall depending on the choice of q. This means that a reasonable measure of quality requires that we typically average together the result of several queries using some likely distribution in the choice of q.

2.3 Properties Affecting Similarity Queries

An embedding induced by a mapping F is said to be *contractive* with respect to S if $\delta(F(o_1), F(o_2)) \leq d(o_1, o_2)$ for all $o_1, o_2 \in S$. In other words, $c_2 = 1$ in (1) and, thus, the distortion is just c_1. Contractiveness of the embedding is a very useful property in similarity search, and many other applications, as it has implications for pruning the search. However, for similarity search, contractiveness with respect to S is not sufficient for achieving 100 percent recall. In particular, query objects are usually not members of S, so contractiveness with respect to S would not say anything about the relationship between $d(q, o)$ and $\delta(F(q), F(o))$ for a query object $q \in \mathbb{U} \setminus S$ and data object $o \in S$. If we knew in advance the set $Q \subset \mathbb{U}$ of all potential query objects, we could define Q/S-contractiveness such that $\delta(F(q), F(o)) \leq d(q, o)$ for any $q \in Q$ and $o \in S$. For example, consider a range query with a radius of r with respect to object $q \in S$, where we would like to identify objects $o \in S$ such that $d(q, o) \leq r$. If the embedding is Q/S-contractive, we are ensured that $d(q, o) > r$ if $\delta(F(q), F(o)) > r$ for any $q \in Q$ and $o \in S$. Thus, we can safely prune all objects o from the search for which $\delta(F(q), F(o)) > r$ without any *false dismissals*—that is, no relevant object is dropped from the query result and, thus, we have 100 percent recall.

Unfortunately, the set Q of all potential query objects is usually not known at the time when the mapping F is constructed, except for the trivial case $Q = \mathbb{U}$. Furthermore, if dynamic updates are made to the database S and F is only updated occasionally (e.g., when the quality of the embedding has deteriorated to a certain level), then the contents of S at query time may be different than when the mapping was constructed. Therefore, for ensuring 100 percent recall for similarity search, it is usually necessary for F to be contractive with respect to $\mathbb{U}$. In the remainder of this paper, we use the term contractive to mean that F is contractive with respect to $\mathbb{U}$, i.e., $\delta(F(o_1), F(o_2)) \leq d(o_1, o_2)$ for all $o_1, o_2 \in \mathbb{U}$.

Another useful, albeit rarely satisfied, property of a mapping F is *proximity preservation*, i.e., the property that $d(o_1, o_2) \leq d(o_1, o_3) \Rightarrow \delta(F(o_1), F(o_2)) \leq \delta(F(o_1), F(o_3))$. If this property holds, we can perform nearest-neighbor queries in the embedding space and be assured that the result is valid in the original space. In other words, if q is a query object and $o \in S$ is its nearest neighbor, i.e., $d(q, o) \leq d(q, o')$ for all objects $o' \in S$, then we know that $F(o)$ is also the nearest neighbor of $F(q)$ with respect to δ. Thus, we can simply perform the nearest-neighbor query using $F(q)$. Since the proximity preservation property is rarely satisfied, it is interesting to ask if we can derive a relaxed version of it from other properties, such as distortion. This is indeed possible and yields the following relaxed form given a distortion of c_1c_2 (see (1)):

$$\begin{aligned} & d(o_1, o_2) \leq d(o_1, o_3) \\ \Rightarrow & 1/c_2 \cdot \delta(F(o_1), F(o_2)) \leq d(o_1, o_2) \leq d(o_1, o_3) \leq \\ & c_1 \cdot \delta(F(o_1), F(o_3)) \\ \Rightarrow & \delta(F(o_1), F(o_2)) \leq c_1c_2 \cdot \delta(F(o_1), F(o_3)). \end{aligned}$$

In other words, $\delta(F(o_1), F(o_2))$ is never larger than $\delta(F(o_1), F(o_3))$ by more than a factor of c_1c_2. Thus, if a nearest-neighbor query is performed in the embedding space, with the result that $F(o')$ with $o' \in S$ is the nearest neighbor of $F(q)$, then $\delta(F(q), F(o'))$ can be smaller than $\delta(F(q), F(o))$ by as much as a factor of c_1c_2, where o is the true nearest neighbor of q in S. Equivalently, $d(q, o')$ may be larger than $d(q, o)$ by a factor as large as c_1c_2. In many applications, exact responses to nearest-neighbor queries are not crucial and approximate responses are satisfactory, at least if the error is not too high (e.g., see [25], [26], [27]). Unfortunately, the worst-case distortion is often fairly high (e.g., $O(\log N)$), so the relaxed proximity preservation property may yield too large an error for nearest-neighbor queries.

Nevertheless, if the mapping F is contractive, efficient nearest-neighbor query algorithms can be implemented that give an exact result.[2] Such algorithms use a filter and refine strategy [6], [28], [29], as described in Section 1.3. In particular, in the "filter" step, the embedding space is used as a filter to produce a set of candidates. The satisfaction of the contractive property makes it possible to guarantee that the correct result is among the candidates. For example, if o is the actual nearest neighbor of the query object q, then the filter step must at the very least produce as candidates all objects o' such that $\delta(F(q), F(o')) \leq d(q, o)$. In the "refine" step, the actual distance must be computed for all the candidates to determine the actual nearest neighbor.

To elaborate on how such a query is implemented, suppose that we want to find the nearest object to a query object q. We first determine the point $F(q)$ corresponding to q. Next, we examine the objects in the order of their distances from $F(q)$ in the embedding space. When using a multidimensional index, this can be achieved by using an incremental nearest neighbor algorithm [30], [31]. Suppose that point $F(a)$ corresponding to object a is the closest point to $F(q)$ at a distance of $\delta(F(a), F(q))$. We compute the distance $d(a, q)$ between the corresponding objects. At this point, we know that any object x with $\delta(F(x), F(q)) > d(a, q)$ cannot be the nearest neighbor of q since the contractive property then guarantees that $d(x, q) > d(a, q)$. Therefore, $d(a, q)$ now serves as an upper bound on the nearest-neighbor search in the embedding space. We now find the next closest point $F(b)$ corresponding to object b, subject to our distance constraint $d(a, q)$. If $d(b, q) < d(a, q)$, then b and $d(b, q)$ replace object a and $d(a, q)$ as the current closest object and upper bound distance, respectively; otherwise, a and $d(a, q)$ are retained. This search continues until encountering a point $F(x)$ with $\delta(F(x), F(q)) > d(y, q)$, where y is the current closest object which is now guaranteed to be the actual closest object to q.

2. Variants that allow a small error in the result are also possible if we wish to trade off accuracy for a possible increase in efficiency.

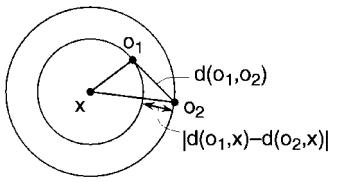

Fig. 1. Demonstration of the distance bound $|d(o_1,x)-d(o_2,x)| \leq d(o_1,o_2)$. The objects o_1, o_2, and x are represented as points and the distances between them by the lengths of the line segments between them.

3 SparseMap

SparseMap is an embedding method based on a powerful class of embedding methods known as Lipschitz embeddings [17], [18]. As originally defined, Lipschitz embeddings are not suitable for similarity search due to the large computational cost. Thus, SparseMap includes heuristics aimed at reducing the cost of producing the embedding. Below, we first describe Lipscitz embeddings (Section 3.1) and then explain the heuristics of SparseMap (Section 3.2).

3.1 Lipschitz Embeddings

In Lipschitz embeddings, a coordinate space is defined such that each axis corresponds to a reference set, drawn from the set S of objects, as described in Section 3.1.1. How to select the reference sets is shown in Section 3.1.2.

3.1.1 Definition

A *Lipschitz embedding* is defined in terms of a set R of subsets of S, $R = \{A_1, A_2, \ldots, A_k\}$. The subsets A_i are termed the *reference sets* of the embedding. Let $d(o, A)$ be an extension of the distance function d to a subset $A \subset S$, such that $d(o, A) = \min_{x \in A}\{d(o,x)\}$. An embedding with respect to R is defined as a mapping F such that $F(o) = (d(o, A_1), d(o, A_2), \ldots, d(o, A_k))$. In other words, what we are doing is defining a coordinate space where each axis corresponds to a subset $A_i \subset S$ of the objects and the coordinate values of object o are the distances from o to the closest element in each of A_i.

The intuition behind the Lipschitz embedding is that, if x is an arbitrary object in the data set S, some information about the distance between two arbitrary objects o_1 and o_2 is obtained with the aid of $d(o_1, x)$ and $d(o_2, x)$, i.e., the value $|d(o_1,x) - d(o_2,x)|$. In particular, due to the triangle inequality, we have $|d(o_1,x) - d(o_2,x)| \leq d(o_1,o_2)$, as illustrated in Fig. 1. This argument can be extended to a subset A: The value $|d(o_1, A) - d(o_2, A)|$ is a lower bound on $d(o_1, o_2)$. This can be seen as follows: Let $x_1, x_2 \in A$ be such that $d(o_1, A) = d(o_1, x_1)$ and $d(o_2, A) = d(o_2, x_2)$. Since $d(o_1, x_1) \leq d(o_1, x_2)$ and $d(o_2, x_2) \leq d(o_2, x_1)$, we have $|d(o_1, A) - d(o_2, A)| = |d(o_1, x_1) - d(o_2, x_2)|$. Accounting for the fact that $d(o_1, x_1) - d(o_2, x_2)$ can be positive or negative, we have

$$|d(o_1,x_1) - d(o_2,x_2)| \leq \max\{|d(o_1,x_1) - d(o_2,x_1)|, |d(o_1,x_2) - d(o_2,x_2)|\}.$$

Finally, from the triangle inequality, we have

$$\max\{|d(o_1,x_2) - d(o_2,x_2)|, |d(o_1,x_1) - d(o_2,x_1)|\} \leq d(o_1,o_2).$$

Thus, $|d(o_1, A) - d(o_2, A)|$ is a lower bound on $d(o_1, o_2)$. By using a set R of subsets, we increase the likelihood that the distance $d(o_1, o_2)$ between two objects o_1 and o_2 (as measured relative to other distances) is captured adequately by the distance in the embedding space between $F(o_1)$ and $F(o_2)$ (i.e., $\delta(F(o_1), F(o_2))$).

3.1.2 Selecting Reference Sets

With a suitable definition of R, the set of reference sets, we can establish bounds on the distance $\delta(F(o_1), F(o_2))$ for all pairs of objects $o_1, o_2 \in S$, where δ is one of the L_p metrics. Such a definition was provided by Linial et al. [1], [32] based, in part, on previous work by Bourgain [17]. In particular, in their definition [1], R consists of $O(\log^2 N)$ randomly selected subsets of S, where each group of $O(\log N)$ subsets is of size 2^i, where $i = 1, \ldots, O(\log N)$; more concretely, the value $O(\log N)$ is typically approximately $\lfloor \log_2 N \rfloor$. Thus, $R = \{A_1, A_2, \ldots, A_k\}$, where $k = \lfloor \log_2 N \rfloor^2$ and A_i is of size 2^j with $j = \lfloor (i-1)/(\log_2 N) + 1 \rfloor$. The embedding proposed by Linial et al. [1] is a variant of the basic Lipschitz embedding, where each coordinate value is divided by a factor that depends on k. In particular, if δ is the L_p metric, F is defined such that $F(o) = (d(o, A_1)/q,\ d(o, A_2)/q, \ldots, d(o, A_k)/q)$, where $q = k^{1/p}$. Given this definition, Linial et al. [1] prove that F satisfies

$$\frac{c}{\lfloor \log_2 N \rfloor} \cdot d(o_1, o_2) \leq \delta(F(o_1), F(o_2)) \leq d(o_1, o_2), \qquad (2)$$

for any pair of objects $o_1, o_2 \in S$, where $c > 0$ is a constant.[3] Thus, the relative amount of deviation of the distance values in the embedding space with respect to the original distance values, known as *distortion*, is guaranteed to be $O(\log N)$ (with high probability). The proof for the bound $c/\lfloor \log_2 N \rfloor d(o_1, o_2) \leq \delta(F(o_1), F(o_2))$ is fairly complex [1], [17], and is beyond the scope of this paper. However, the bound $\delta(F(o_1), F(o_2)) \leq d(o_1, o_2)$ is easy to show. In particular, for each $A_i \in R$, we have $|d(o_1, A_i) - d(o_2, A_i)| \leq d(o_1, o_2)$, as shown in Section 3.1.1. Thus, when δ is an arbitrary L_p distance metric,

$$\begin{aligned} \delta(F(o_1), F(o_2)) &= \left(\sum_{i=1}^{k} \left(\frac{d(o_1, A_i) - d(o_2, A_i)}{k^{1/p}} \right)^p \right)^{1/p} \\ &\leq \left(k \cdot \frac{d(o_1, o_2)^p}{k} \right)^{1/p} = d(o_1, o_2). \end{aligned} \qquad (3)$$

A distortion of $O(\log N)$ may be too large to effectively preserve distances. For example, if the range of distance values is less than the distortion, then the relative order of the neighbors of a given object may be completely scrambled. However, note that $O(\log N)$ is a worst-case (probabilistic) bound. In many cases, the actual behavior is much better. For example, in a computational biology application [21], [33], the embedding defined above was found to lead to good preservation of clusters, as defined by biological functions of proteins.

Unfortunately, the embedding of [1] described above is rather impractical for similarity searching for two reasons. First, due to the number and sizes of the subsets in R, there is a high probability that all N objects appear in some set in R. Thus, when computing the embedding $F(q)$ for a query object q (which generally is not in S), we would need to compute the distances between q and practically all objects

3. More accurately, since the sets A_i are chosen at random, the proof is probabilistic and c is a constant with high probability.

in S, which is exactly what we wish to avoid. Second, the number $\lfloor \log_2 N \rfloor^2$ of subsets in R and, thus, the number of coordinate values (i.e., dimensions) in the embedding, is rather large. Even with as few as 100 objects, the number of dimensions is 36, which is much too high to index on efficiently with multidimensional indexing methods. These drawbacks were acknowledged in [1], but addressing them was left for future work (the only suggestion that was made was to drop the sets A_i of largest sizes).

3.1.3 Related Work

A number of researchers have presented variations on Lipschitz embeddings as described above. Cowen and Priebe [34] presented a method where the number and sizes of the reference sets A_i are chosen based on an objective function that is meant to capture the quality of clustering that results. Faragó et al. [35] and Vleugels and Veltkamp [36] independently suggested using singleton reference sets, each of whose sole member is termed a *vantage object* in [36]. Farago et al. [35] were primarily concerned with properties of the search and do not explicitly specify how to choose the "vantage objects," but Vleugels and Veltkamp [36] suggest picking them at random from the data set (as is the case with singleton reference sets in the method of Linial et al. [1]).

Similarity search methods based on distance matrices [37], [38], [39], [40], [41], also have some relation to Lipschitz embeddings. These methods typically (e.g., see [38]) make use of a matrix $A = (a_{ij})$ of distances, where $a_{ij} = d(o_i, p_j)$ and $T = \{p_1, p_2, \ldots, p_k\}$ is a set of k reference objects; for some methods (e.g., [40]), $T = S$ and $k = N$). Thus, the row vectors of A correspond to the result of a Lipschitz embedding using the singleton reference sets of objects in T. However, the search algorithms proposed for distance matrix methods do not explicitly treat the row vectors as if they represent points in geometric space (i.e., by using the Euclidean metric, or some other Minkowski metric).

3.2 SparseMap Heuristics

SparseMap was originally proposed for mapping a database of proteins into Euclidean space and is built on the work of Linial et al. [1] in that the same set of reference sets R is used. The SparseMap method [21] comprises two heuristics, each aimed at alleviating one of the drawbacks discussed in Section 3.1.2—that is, the potentially high cost of computing the embedding in terms of the number of distance computations that are needed, and the large number of coordinate values. The first heuristic reduces the number of distance computations by calculating an upper bound $\hat{d}(o, A_i)$ instead of the exact value $d(o, A_i)$, while the second heuristic reduces the number of dimensions by using a "high quality" subset of R instead of the entire set as defined in Section 3.1.2. Both heuristics have the potential of reducing the quality of the embedding, in terms of the correspondence of distances in the original metric space and in the embedding space, but their goal [21] is to maintain the quality to the greatest extent possible. Note that the embedding used in SparseMap employs the regular Lipschitz embedding with respect to R, rather than the embedding proposed in [1] (which divides the distances $d(o, A_i)$ by $k^{1/p}$) and uses the Euclidean distance metric.

A drawback of SparseMap is that the resulting embedding cannot be shown to satisfy any constraints on the distortion. In other words, when the SparseMap method is applied, $d_E(F(o_1), F(o_2))$ can be arbitrarily smaller or larger than $d(o_1, o_2)$. In Section 3.3, we address the question of whether this shortcoming can be rectified, especially in terms of the contractive property.

In SparseMap, the coordinate values of the vectors are computed one by one. In other words, if $R = \{A_1, A_2, \ldots, A_k\}$ is the sequence of reference sets in order of size, we first compute $\hat{d}(o, A_1)$ for all objects $o \in S$, next $\hat{d}(o, A_2)$ for all objects o, etc., where $\hat{d}$ denotes the heuristic upper bound on d. The heuristic for computing $\hat{d}$ exploits the partial vector that has already been computed for each object and calculates only a fixed number of distance values for each object (as opposed to $|A_i|$ distance values). In particular, for each object $x \in A_i$, it computes $d_E(F_{i-1}(o), F_{i-1}(x))$, where F_{i-1} is the embedding based on $A_1, \ldots, A_{i-1}$. On the basis of this approximate distance value, a fixed number l of objects in A_i having the smallest approximate distance value from o is picked and the actual distance value $d(o, x)$ for each such object x is computed. The smallest distance value among those serves as the upper-bound distance value $\hat{d}(o, A_i)$, which becomes the ith coordinate value of the vector corresponding to o in the embedding.

The second heuristic involved in SparseMap reduces the dimensionality of the result and is termed *greedy resampling* in [21]. Greedy resampling is applied after the k coordinate axes have all been determined, and its goal is to reduce the number of coordinate axes to $k' < k$, thus eliminating some of the reference sets A_i. A natural question is whether we cannot eliminate a poor reference set A_i before computing all the approximate distances $\hat{d}(o, A_i)$. However, the problem is that we cannot know whether or not a set A_i is good before evaluating $\hat{d}(o, A_i)$ (or $d(o, A_i)$) for each object o. The basic idea of greedy resampling is to start with a single "good" coordinate axis and then incrementally add coordinate axes that maintain "goodness." In particular, initially, the coordinate axis whose sole use leads to the least stress [22] is determined (this is somewhat analogous in spirit to basing the first coordinate axis on a pair of objects that are far apart in the FastMap method as described in Section 4). Unfortunately, calculating the stress requires computing distances for all pairs of objects, which is prohibitively expensive. Instead, the heuristic computes the stress based on some fixed number of object pairs (e.g., 4,000 in experiments in [21], which constituted 10 percent of the total number of pairs). Next, the coordinate axis that leads to the least stress when used in conjunction with the first axis is determined. This procedure is continued until the desired number of coordinate axes has been obtained.

In order to study the validity of the SparseMap method, various experiments are presented in [21], in which the data sets consist of proteins (or more accurately, the amino acid sequences that comprise each protein). The focus of the presentation is mainly on comparing the performance of SparseMap with that of FastMap [20], another embedding method proposed for similarity searching and described in greater detail in Section 4. Both methods are based on heuristics where some parameter controls the number of distance computations that are performed. In SparseMap, this is the number of actual distance computations performed in evaluating $\hat{d}(o, A_i)$, while in FastMap it is the number of iterations performed in the pivot-finding process (see Section 4.3). Thus, the two methods can be made to perform approximately the same number of distance

computations when obtaining a given number of coordinate axes. In the experiments reported in [21], when this is done, the embedding produced by SparseMap is of higher quality than that produced by FastMap in terms of stress as well as cluster preservation (i.e., CPR as defined in Section 2.2 where the clusters were determined using the biological functions of the proteins). This was especially true when the number of coordinate axes was low, which is an important consideration since multidimensional indexing is more effective in low dimensions. In addition, SparseMap was found to scale up better than FastMap, in terms of the time to perform the mapping, as the manner in which the database is accessed leads to fewer disk I/Os.

3.3 Making SparseMap Contractive

A drawback of the embedding that forms the basis of SparseMap [21] (i.e., the regular Lipschitz embedding on the reference sets, without taking the heuristics into account) is that it is not contractive and, thus, does not allow achieving 100 percent recall in similarity queries. In particular, the distance value in the embedding may be as much as a factor of $\log_2 N$ larger than the actual distance value. Two methods can be applied to obtain a contractive embedding. First, the embedding proposed in [1] (where the coordinate values are divided by $k^{1/p}$) can be employed, which is indeed contractive. Second, the distance function δ can be modified to yield the same effect. In particular, if $d_p(F(o_1), F(o_2))$ is one of the Minkowski metrics, we can define $\delta(F(o_1), F(o_1)) = d_p(F(o_1), F(o_2))/(k^{1/p})$. The advantage of modifying the distance function δ rather than the embedding itself is that it allows modifying the number of coordinate axes (for example, during the construction of the embedding and in the second SparseMap heuristic), without changing existing coordinate values. With either method, the embedding would satisfy (2) for any distance metric L_p (i.e., subject to modification when using the second method).

Unfortunately, the heuristics applied in SparseMap do not allow deriving any practical bounds on the distortion resulting from the embedding. In particular, the first heuristic can lead to larger distances in the embedding space, thus possibly causing the contractive property to be violated (in contrast, the second heuristic can only reduce distances in the embedding space). This is because the value of $|\hat{d}(o_1, A_i) - \hat{d}(o_2, A_i)|$ may not necessarily be a lower bound on $d(o_1, o_2)$. To see why, note that the upper bound distances $\hat{d}(o_1, A_i)$ and $\hat{d}(o_2, A_i)$ can exceed the actual distances $d(o_1, A_i)$ and $d(o_2, A_i)$ by an arbitrary amount. In particular, we cannot rule out a situation where $\hat{d}(o_1, A_i) > \hat{d}(o_2, A_i) + d(o_1, o_2)$, in which case $|\hat{d}(o_1, A_i) - \hat{d}(o_2, A_i)| > d(o_1, o_2)$.

Thus, we see that, in order to be able to satisfy the contractive property, we must use the actual values $d(o, A_i)$ in the embedding, rather than an upper bound on these distances as done in SparseMap. Fortunately, there is a way to modify the first heuristic of SparseMap so that it computes the actual value $d(o, A_i)$, while still (at least potentially) reducing the number of distance computations. We illustrate this for the case when the Chessboard distance metric, d_M, is used as δ in the embedding space. Recall that $d_M(F(o_1), F(o_2)) \leq d(o_1, o_2)$ as shown in Section 3.1.2 (the key observation is that $|d(o_1, A_i) - d(o_2, A_i)| \leq d(o_1, o_2)$ for all A_i). Furthermore, if F_i is the partial embedding for the first i coordinate values, we also have $d_M(F_i(o_1), F_i(o_2)) \leq d(o_1, o_2)$. In this modified heuristic for computing $d(o, A_i)$, instead of computing the actual distance value $d(o, x)$ for only a fixed number of objects $x \in A_i$, we must do so for a variable number of objects in A_i. In particular, we first compute the approximate distances $d_M(F_{i-1}(o), F_{i-1}(x))$ for all objects $x \in A_i$, which are lower bounds on the actual distance value $d(o, x)$. Next, we compute the actual distances of the objects $x \in A_i$ in increasing order of their lower bound distances, $d_M(F_{i-1}(o), F_{i-1}(x))$. Let $y \in A_i$ be the object whose actual distance value $d(o, y)$ is the smallest distance value so far computed following this procedure. Once the lower bound distances $d_M(F_{i-1}(o), F_{i-1}(x))$ of all remaining elements $x \in A_i$ are greater than $d(o, y)$, we are assured that $d(o, A_i) = d(o, y)$.

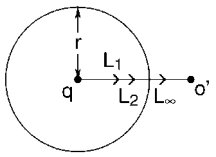

Fig. 2. A hypothetical range query example where an object o' is outside the distance range r from the query object q. The distance $\delta(F(q), F(o'))$ from q to o' in the embedding space will lie somewhere on the line between q and o'. Thus, this distance may lie inside the query range if δ is based on L_1 or L_2, but outside it if δ is based on L_∞.

Even though we described our modified heuristic in terms of the Chessboard distance metric, by using a suitable definition of the distance function δ the heuristic can be applied to any Minkowski metric L_p. In particular, if k' is the current number of coordinate axes (at the completion of the process, $k' = k$), the distance function δ based on L_p is defined as

$$\delta(F_{k'}(o_1), F_{k'}(o_2)) = \frac{(\sum_i |d(o_1, A_i) - d(o_2, A_i)|^p)^{1/p}}{(k')^{1/p}}. \quad (4)$$

For any choice of p, this distance metric makes F contractive; note the similarity with (3). Moreover, observe that, for fixed values of o_1, o_2, and $F_{k'}$, the function δ defined by (4) increases with increasing values of p. For example, for $p = 1$, $\delta(F_{k'}(o_1), F_{k'}(o_2))$ is the average among the coordinate value differences, while for $p = \infty$, it is the maximum difference. Thus, the use of the Chessboard metric L_∞ would lead to the largest values of $\delta(F_{k'}(o_1), F_{k'}(o_2))$ for any given choice of the sets A_i. For similarity queries, given a fixed set of reference sets A_i, this would therefore lead to the best possible pruning during search, as well as for the modified heuristic described above. To see why this is the case, suppose that we are performing a range query with query object q and query radius r, and we wish to report all objects o such that $d(q, o) \leq r$. Let o' be an object that is too far from q, i.e., $d(q, o') > r$. However, if $\delta(F(q), F(o')) \leq r$, o' will be a part of the result set when performing a query in the embedding space. Thus, the situation can easily arise that $\delta(F(q), F(o')) \leq r$ when basing δ on the City Block or Euclidean distance metrics (i.e., L_1 or L_2), but $\delta(F(q), F(o')) > r$ when δ is based on the Chessboard distance metric L_∞. Such a hypothetical example is illustrated in Fig. 2.

Although the modified heuristic presented above will likely lead to a higher number of distance computations than the SparseMap heuristic, the higher cost of the embedding (which mainly affects preprocessing) may be justified, as the

resulting embedding is contractive. This allows effective pruning in similarity queries, while obtaining accurate results, as we get 100 percent recall and, thus, do not miss any relevant answers.

4 FastMap

FastMap [20] is a general embedding technique that is inspired by dimensionality reduction methods for Euclidean space based on linear transformations such as the Karhunen-Loève transform (KLT) [10]. However, as we show in Section 5, it is only guaranteed to be contractive for the Euclidean distance metric. In this section, we first review the motivation for the development of FastMap (Section 4.1). Next, we outline how it works (Sections 4.2, 4.3, 4.4, 4.5, and 4.6). In particular, Section 4.2 explains the general principles behind FastMap by outlining how the coordinate axes that make up the mapping are constructed. This includes the introduction of a modified distance function. Section 4.3 describes how the pivot objects that anchor the lines that form the newly formed coordinate axes are chosen. Section 4.4 shows how the first coordinate value is determined. Section 4.5 presents the modified distance function for computing the remaining coordinate values, while Section 4.6 applies this modified distance function to actually compute the remaining coordinate values.

4.1 Motivation

The Karhunen-Loève transform (KLT) [10] is a linear transformation that allows determining coordinate axes that retain as much distance information as possible; it is essentially equivalent to Principal Component Analysis (PCA) and both can be computed using Singular Value Decomposition (SVD, see Section 6.2). In particular, for a set S of points in an m-dimensional Euclidean space, KLT identifies a new set of m coordinate axes, represented by an orthonormal set $V = \{v_1, v_2, \ldots, v_m\}$ of basis vectors (i.e., each basis vector has a length of 1 and any two basis vectors are orthogonal). The origin of the new coordinate system is taken to be the center of gravity of the points in S and the new coordinate values are obtained by projecting each point in S onto the basis vectors.

Loosely speaking, the set V is chosen such that the variance, when projecting S on the basis vectors, is as great as possible along each basis vector in turn (the variance indicates the level of "spread" when projecting S on a basis vector). The implication is that $d_k(F_k(p_1), F_k(p_2))$ will preserve distances as much as possible, in a mean-square sense, if we denote by $F_k(p)$ the projection of a point p onto the first k basis vectors in V and by d_k the Euclidean distance in k-dimensional Euclidean space. In other words, KLT results in the linear transformation that minimizes

$$\sum_{p_1,p_2} (d_k(F_k(o_1), F_k(o_2)) - d(o_1, o_2))^2.$$

Thus, the mapping F_k provides dimension-reduction to a k-dimensional Euclidean space that provides good distance-preservation among the points in S. However, KLT is not applicable when using distance metrics other than the Euclidean, let alone when S is not drawn from a vector space.

FastMap is an attempt to generalize the principles of dimension-reduction methods based on linear transformations for obtaining embeddings of arbitrary metric spaces (rather than just for Euclidean spaces) into k-dimensional Euclidean space. Nevertheless, as we shall see in the remainder of this section as well as in Section 5, this generalization is not without its limitations (i.e., the resulting mapping is no longer guaranteed to be contractive). Besides the motivation of generalizing KLT to arbitrary metric spaces, FastMap is designed to be faster than KLT, which takes $O(N \cdot m^2)$ time. In contrast, FastMap requires $O(N \cdot k)$ distance computations, each of which is $O(m)$ assuming that we start out with m-dimensional vector data. Thus, in this setting, a more accurate assessment of the execution time complexity of the FastMap method is $O(Nmk)$.

4.2 General Principles

In the following, we explain how the FastMap method works in some detail. Many of the derivations used in the development of the method make an implicit assumption that (S, d) is a subset of a Euclidean space of some dimensionality, or in other words, that d is the Euclidean distance metric.[4] Nevertheless, FastMap can be applied with varying success with other distance metrics. In particular, as we will see in Section 5, use of the FastMap method with other distance metrics will often mean that some desirable key aspects such as the contractive property will not necessarily hold nor will we always be able to obtain as many as k coordinate axes (even as few as 1).[5] Similarly, due to the nature of the FastMap method, the best result is obtained when δ, the distance function in the embedding space, is the Euclidean distance metric. Thus, unless otherwise stated, we assume δ to be the Euclidean distance metric.

The FastMap method works by imagining that the objects are points in a hypothetical high-dimensional Euclidean space of unknown dimension—that is, a vector space with the Euclidean distance metric. However, the various implications of this Euclidean space assumption are not explored by Faloutsos and Lin [20] in their development of the method. In the sequel, the terminology reflects the assumption that the objects are points (e.g., a line can be defined by two objects, etc.). The coordinate values corresponding to these points are obtained by projecting them on k mutually orthogonal directions, thereby forming the coordinate axes of the space in which the points are embedded. The projections are computed using the given distance function d. The coordinate axes are constructed one by one, where at each iteration, two objects (termed *pivot objects*) are chosen, a line is drawn between them that serves as the coordinate axis, and the coordinate value along this axis for each object o is determined by mapping (i.e., projecting) o onto this line.

Assume, for the sake of discussion, that the objects are actually points (this makes it easier to draw the examples that we use in our explanation) and that they lie in an m-dimensional space. We prepare for the next iteration (where the next coordinate axis established) by determining the $(m-1)$-dimensional hyperplane H perpendicular to the line that forms the previous coordinate axis, and projecting all of the objects onto H. The projection is performed by defining a new distance function d_H that measures the distance between the projections of the objects on H. In particular, we will see that

4. More precisely, the assumptions made by FastMap are valid if (S, d) is isometric to a subset of some Euclidean space. Below, when we say that a property applies (or not) when d is a Euclidean metric, we also mean that it applies (or not) in this isometric case.

5. Of course, this would not be a problem if the "intrinsic" dimensionality [42] of the data were low, but this need not be the case.

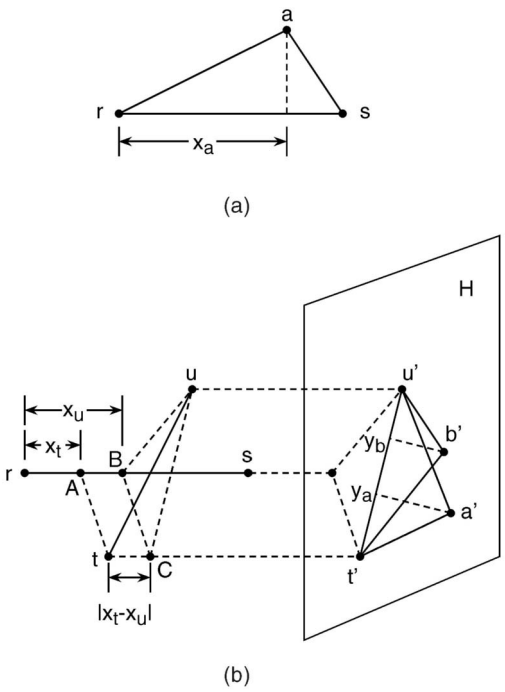

Fig. 3. Examples of projections of objects with the FastMap method on (a) the first coordinate axis and (b) the second coordinate axis.

d_H is derived from the original distance function d and the coordinate axes determined so far. At this point, the problem has been replaced by a recursive variant of the original problem with m and k reduced by one and a new distance function d_H. This process is continued until the necessary number of coordinate axes has been determined.

Fig. 3 illustrates how the first coordinate axis is determined, how the objects are mapped onto the hyperplane H, and how the projected objects are used to determine the second coordinate axis. Fig. 3a shows the result of the projection that yields the first coordinate axis where the pivot objects are r and s. In particular, the first coordinate value x_a for object a (i.e., the first coordinate value in the vector $F(a)$) is the distance from r to the projection of a onto the line through r and s. We postpone for now the discussion of Fig. 3b, which illustrates how d_H and the next set of coordinate values is determined.

4.3 Choosing Pivot Objects

As we saw, the pivot objects that are chosen at each step serve to anchor the line that forms the newly formed coordinate axis. When projecting the remaining objects on this line, the values should ideally be well spread out since this generally means that more distance information can be extracted from the projected values—that is, for any pair of objects a and b, it is more likely that $|x_a - x_b|$ is large, thereby providing more information. Recall that the KLT method, described in Section 4.1, uses variance as a measure of spread. However, to reduce the computational cost, FastMap resorts to the weaker notion of *range* as a measure of spread, $\max_{a,b \in S} |x_a - x_b|$, and approximates the range with the distance between the pivot objects. Thus, in each iteration, FastMap attempts to identify a pair of pivot objects that are far away from each other. Unfortunately, determining the farthest pair of objects in a given set of N objects requires $O(N^2)$ distance computations which is prohibitively expensive.

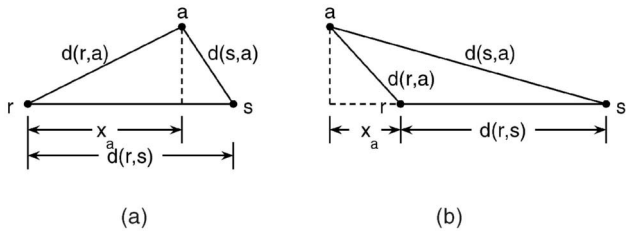

Fig. 4. Examples of two possible positions for the projection of an object on the line joining the points corresponding to the pivot objects.

Faloutsos and Lin [20] propose a heuristic for computing an approximation of the farthest pair of objects. This heuristic first arbitrarily chooses one of the objects a. Next, it finds the object r which is farthest from a. Finally, it finds the object s which is farthest from r. The last step can be iterated a number of times (e.g., five times [20]) in order to obtain a better estimate of the pair that is farthest apart. In fact, it can be shown that, for a given set of N objects, the procedure for finding the farthest pair of objects can be iterated a maximum of $N-1$ steps for a total of $O(N^2)$ distance computations. The heuristic process of finding the pivot objects requires $O(N)$ distance computations as long as the number of iterations is fixed. Unfortunately, the $O(N)$ cost bound for the heuristic may not always hold, as shown in Section 5. Note that the original distance function d is used only when determining the first coordinate axis. However, the modified distance functions (resulting from successive projections on hyperplanes) used for subsequent coordinate axes are based on d and, thus, an evaluation of d is also required for any distance computations in later steps.

Choosing pivot objects r and s in this way guarantees that the distance between the actual farthest pair, a and b, is at most $2d(r,s)$; i.e., $d(r,s) \geq \frac{1}{2} d(a,b)$ (see [24] for a proof). However, this bound is guaranteed to hold only for the first pair of pivot objects, as shown in Section 5, since it is possible that the distance functions used to determine the second and subsequent coordinate values do not satisfy the triangle inequality.

4.4 Deriving the First Coordinate Value

In order to understand better how and why the FastMap mapping process works, let us examine its mechanics in greater detail as we compute the first coordinate value. Initially, we project the objects on a line between the pivot objects, say r and s, as shown in Fig. 4 for an object a. Note that the projection of object a may actually lie beyond the line segment between r and s as shown in Fig. 4b. This does not pose problems, but may cause x_a to be negative. The actual value of x_a is obtained by solving the following equation for x_a:

$$d(r,a)^2 - x_a^2 = d(s,a)^2 - (d(r,s) - x_a)^2. \tag{5}$$

Expanding terms in (5) and rearranging yields

$$x_a = \frac{d(r,a)^2 + d(r,s)^2 - d(s,a)^2}{2d(r,s)}. \tag{6}$$

Observe that (5) is obtained by applying the Pythagorean theorem to each half of the triangle in Fig. 4a (a similar interpretation applies to the case in Fig. 4b). Since the Pythagorean theorem is specific to Euclidean space, we have here an instance where Faloutsos and Lin [20] in their development of the method make the implicit assumption that d is the Euclidean distance metric. Thus, the equation is

only a heuristic when used for general metric spaces. (The implications of this heuristic are explored in Section 5; namely, we find that the embedding produced by FastMap is not contractive and this may cause the mapping process to terminate prematurely.)

A number of observations can be made about x_a, based on (6) and the selection of pivot objects. First, $x_r = 0$ and $x_s = d(r, s)$, as would be expected. Second, note that $|x_a| \leq d(r, s)$, implying that the range (as defined above) along the first coordinate axis is at most $2d(r, s)$. In fact, it can be shown that the range is never larger than the distance between the farthest pair of objects (see [24]), which is at most $2d(r, s)$ as mentioned in Section 4.3. Thus, the range obtained from pivots r and s is at least half of the maximum obtainable range, since the range is at least $d(r, s)$ when r and s serve as the pivot objects (as $x_r = 0$ and $x_s = d(r, s)$). Unfortunately, as we show in Section 5, it is possible that the distance functions used to determine the second and subsequent coordinate values do not satisfy the triangle inequality; in which case, the above bounds may not hold.

4.5 Projected Distance

Before we can determine the second coordinate value for each object, we must derive d_H, the distance function for the distances between objects when projected onto the hyperplane H, as mentioned in Section 4.2. Fig. 3b illustrates how the objects are projected onto the hyperplane H and how the projected objects are used to determine the second coordinate axis. For expository purposes, assume that the underlying space is three-dimensional. In this case, points A and B are the projections of objects t and u, respectively, on the first coordinate axis (formed by the line joining the pivot objects r and s) with a separation of $|x_t - x_u|$. Points t' and u' are the projections of objects t and u, respectively, on the plane H that is perpendicular to the line between r and s that forms the first coordinate axis. Point C is the projection of u onto the line through t and t', parallel to the line through r and s. Thus, the distance between t' and u' equals the distance between C and u. The latter can be determined by applying the Pythagorean theorem since the angle at C in the triangle tuC is $90°$. Therefore, we have

$$d(t, u)^2 = d(t, C)^2 + d(C, u)^2 = (x_t - x_u)^2 + d(t', u')^2. \quad (7)$$

Thus, defining $d_H(t, u) = d(t', u')$ and changing the order of the terms, we obtain

$$d_H(t, u)^2 = d(t, u)^2 - (x_t - x_u)^2. \quad (8)$$

Note that (8) applies to any pair of objects t and u and not just to the ones that serve as pivots in the next iteration, as is the case in Fig. 3b. Also, note that Faloutsos and Lin [20] use the notation $d_H(t', u')$ rather than $d_H(t, u)$.

Observe that this is another occasion where Faloutsos and Lin [20] in their development of the method make the implicit assumption that d is the Euclidean distance metric (or that (S, d) is isometric to a subset of some Euclidean space). As pointed out before, this assumption has some undesirable side-effects. For example, as we show in Section 5, if d is not a Euclidean distance metric, d_H may fail to satisfy the triangle inequality, which in turn may cause (6) to produce coordinate values that violate the contractive property. Furthermore, violation of the contractive property in earlier iterations of FastMap may cause negative values of $d_H(a, b)^2$. This complicates the search for pivot objects, as the square root of a negative value is a complex number, which in this case means that a and b (or, more precisely, their projections) cannot serve as pivot objects.

4.6 Subsequent Iterations

Each time we recursively invoke the FastMap coordinate determination method, we must determine the distance function d_H for the current set of projections in terms of the current distance function (i.e., the one that was created in the previous recursive invocation). Thus, the original distance function d is only used when obtaining the first coordinate axis. In subsequent iterations, d is the distance function d_H from the previous iteration. At this point, it is instructive to generalize (6) and (8) to yield a recursive definition of the distance functions and the resulting coordinate values for each object. Before we do so, we must define a number of symbols, each representing the ith iteration of FastMap. In particular, x_o^i is the ith coordinate value obtained for object o, $F_i(o) = \{x_o^1, x_o^2, \ldots, x_o^i\}$ denotes the first i coordinate values of $F(o)$, d_i is the distance function used in the ith iteration, and p_1^i and p_2^i denote the two pivot objects chosen in iteration i (with the understanding that p_2^i is the farthest object from p_1^i). Now, the general form of (6) for iteration i is

$$x_o^i = \frac{d_i(p_1^i, o)^2 + d_i(p_1^i, p_2^i)^2 - d_i(p_2^i, o)^2}{2d_i(p_1^i, p_2^i)}, \quad (9)$$

given the recursive distance function definition

$$\begin{aligned} d_1(a, b) &= d(a, b) \quad (10)\\ d_i(a, b)^2 &= d_{i-1}(a, b)^2 - (x_a^{i-1} - x_b^{i-1})^2 \\ &= d(a, b)^2 - d_E(F_{i-1}(a), F_{i-1}(b))^2. \end{aligned}$$

The process of mapping the N objects to points in a k-dimensional space makes $O(k \cdot N)$ distance computations, as there are $O(N)$ distance calculations at each of k iterations. It requires $O(k \cdot N)$ space to record the k coordinate values of each of the points corresponding to the N objects. It also requires a $2 \times k$ array to record the identities of the k pairs of pivot objects, as this information is needed to process queries. Note that query objects are transformed to k-dimensional points by applying the same algorithm that was used to construct the points corresponding to the original objects, except that we use the existing pivot objects. In other words, given query object q, we obtain its k-dimensional coordinate values by projecting q on the lines formed by the corresponding pivot objects using the appropriate distance function. This process is facilitated by recording the distance between the points corresponding to the pivot objects so, that it need not be recomputed for each query; although this can be done, if we do not want to store these distance values. The entire process of obtaining the k-dimensional point corresponding to the query object takes $O(k)$ distance computations; thus, it is independent of the size N of the database.

4.7 Heuristic for Non-Euclidean Metrics

When d is not a Euclidean metric, the value $d_H(t, u)^2$ in (8) may be negative, as mentioned above. More generally, for the formulation in (10), this implies that $d_i(a, b)^2$ may be negative for $i \geq 2$. Such a situation is undesirable since it means that

$d_i(a,b)$ becomes complex-valued, which precludes the choice of a and b as the pair of pivot objects in iteration i of FastMap. Furthermore, $d_i(a,b)^2$ with a large negative values can cause a large distortion in the distances between coordinate values determined by (9); see Section 5.3.

Wang et al. [23] introduce a heuristic to alleviate this situation. The heuristic defines $d_i(a,b)$, $i \geq 2$, in such a way that it is always real-valued, but possibly negative:

$$d_i(a,b) = \begin{cases} \sqrt{\Delta_i(a,b)} & \text{if } \Delta_i(a,b) \geq 0, \\ -\sqrt{-\Delta_i(a,b)} & \text{otherwise,} \end{cases} \quad (11)$$

where $\Delta_i(a,b) = d_{i-1}(a,b)^2 - (x_a^{i-1} - x_b^{i-1})^2$; that is, $d_i(a,b)^2$ as defined in (10). Although this heuristic apparently resolves the drawbacks of negative $d_i(a,b)^2$ values, it does not correct the fundamental problem with (10), namely, the fact that d_i may violate the triangle inequality if d is not the Euclidean distance metric. Furthermore, notice that this formulation also means that, if $d_i(a,b)$ is negative for some i, then $d_j(a,b)^2 \neq d(a,b)^2 - d_E(F_{j-1}(a), F_{j-1}(b))^2$ for all $j \geq i$.

Notice that, when $d_i(a,b)$ is defined according to (11), the value of $d_i(a,b)^2$ is always nonnegative, regardless of whether $d_i(a,b)$ is negative or not. Thus, in situations where (10) leads to negative values, the coordinate value x_o^i for an object o as determined by (9) can be different depending on which definitions of d_i is used—that is, (10) or (11). If we focus on the result of just one iteration of FastMap, neither definition of d_i is always better than the other, in terms of how well distances are preserved. In particular, it is sometimes better to use (10) and sometimes better to use (11). However, the advantage of the definition in (11) is that the value of $d_i(a,b)^2$ tends to decrease as i increases (i.e., as more iterations are performed). In particular, (11) implies that $d_i(a,b)^2 = |d_{i-1}(a,b)^2 - (x_a^{i-1} - x_b^{i-1})^2|$, so, $d_i(a,b)^2$ is only larger than $d_{i-1}(a,b)^2$ if $(x_a^{i-1} - x_b^{i-1})^2 > 2d_{i-1}(a,b)^2$. In contrast, according to (10), the value of $d_i(a,b)^2$ is monotonically nonincreasing in i, so it can become a large negative value (which has adverse effects on search performance as the embedding is not contractive). In Section 5.3, we explore further the implications of these properties.

With the heuristic described above, two objects a and b can be used as a pivot pair even when $d_i(a,b)$ is negative. In contrast, when using (10), such pairs cannot be utilized. However, it is not clear how appropriate such a choice of pivot objects is in terms of resulting in good distance preservation. In particular, the fact that $d_i(a,b)$ is negative implies that the distance between $F_{i-1}(a)$ and $F_{i-1}(b)$ is greater than $d(a,b)$, so using a and b as pivot objects further increases the amount by which the distance in the embedding space exceeds the distance in the original space (i.e., expansion as defined in Section 5.3).

5 FastMap: Properties

When (S,d) is actually a subset of some Euclidean space, then the embedding F produced by FastMap is always contractive (Section 5.1). Unfortunately, if (S,d) is a finite metric space of any other type, F is not guaranteed to be contractive (Section 5.2). Moreover, the distortion in the distances, as defined in Section 2.2, can be very high. In other words, the distances in the embedding space can be much smaller or much larger than in the original space (Section 5.3). Not only is the lack of the contractive property undesirable for similarity searching purposes, but as we point out, it can also severely degrade the performance of FastMap. In particular, $\Omega(N^2)$ distance computations may now be needed to find an appropriate pair of pivot objects, instead of $O(N)$ which is the case when the contractive property is satisfied. A more detailed discussion of these results can be found in [24].

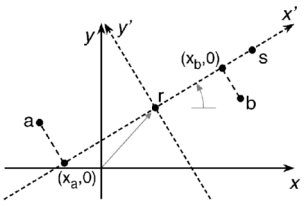

Fig. 5. An example of the rotation and translation of the coordinate system in two-dimensional Euclidean space. In the new coordinate system, the coordinate values of the projected images of data objects a and b become $(x_a, 0)$ and $(x_b, 0)$, respectively.

5.1 Contractiveness for Euclidean Spaces

It should come as no surprise that FastMap produces a contractive embedding when (S,d) is a subset of a Euclidean space (or isometric to one), being that key aspects of FastMap are based on a property unique to Euclidean spaces, namely, the Pythagorean theorem. In particular, both (6), which computes a single coordinate value and (8), which computes the projected distance d_H used in the next iteration, are based on the Pythagorean theorem. Equivalently, FastMap can be seen to be based on the property of the Euclidean metric being invariant under rotation and translation.

Fig. 5 illustrates the rotation and translation of the coordinate system that results from the first iteration of FastMap in two-dimensional Euclidean space. In the second iteration, the distances of the data points are based on their projection on the new axis y', which trivially amounts to a one-dimensional Euclidean space. In general, for m-dimensional Euclidean space, the distance function d_i used in iteration i of FastMap, as defined by (10), corresponds to the Euclidean distance in a $m-i+1$-dimensional Euclidean space. Thus, each iteration of FastMap effectively reduces the dimensionality by one, so that after k iterations, the "residual" distances $d_{k+1}(a,b)$ reflect a $m-k$-dimensional space. Since $d_{k+1}(a,b)^2 = d(a,b)^2 - d_E(F(a),F(b))^2$ and $d_{k+1}(a,b)^2 \geq 0$, we have $d(a,b)^2 \geq d_E(F(a),F(b))^2$, so F is contractive. Furthermore, when $k=m$, $d_{k+1}(a,b) = 0$ for all objects $a,b \in S$, so FastMap can always yield a distance-preserving embedding given a sufficient number of iterations. A complete proof that roughly follows the outline sketched above is found in [24].

5.2 Noncontractiveness of FastMap

Fig. 6 illustrates how FastMap maps objects a and b after having determined just one coordinate value, assuming that r and s are the pivot objects. Based on the figure, it may seem intuitively obvious that the distance between $F(a)$ and $F(b)$ is smaller than that between a and b. Unfortunately, intuition is misleading here, as it is colored by the fact that we perceive the three-dimensional world around us as obeying Euclidean geometry. In particular, the relative lengths of the line

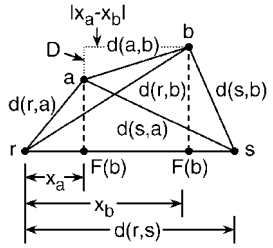

Fig. 6. Example projection of two objects on the line joining the points corresponding to the pivot objects r and s.

segments between the points in Fig. 6 can only arise if d is the Euclidean distance metric. Thus, we see that, in the figure, $d(a,b)^2 = (x_a - x_b)^2 + D^2$ (according to the Pythagorean theorem), so we clearly have $d(a,b) \geq |x_a - x_b|$, meaning that the mapping is contractive. In general, though, we cannot assume that this relationship holds: D^2 may actually be negative! Furthermore, even when the embedding F produced by FastMap happens to be contractive with respect to S, it may not be contractive for query objects drawn from $\mathbb{U} \setminus S$ (i.e., it may not be Q/S-contractive, as defined in Section 2.3).

When (S, d) is not drawn from a Euclidean space, the contractive property may be violated as a result of both (6) and (8), due to the implicit assumption made in deriving these equations on properties unique to Euclidean spaces (i.e., distances being invariant to rotations). This can readily be shown with counterexamples [24]. In particular, in a single iteration of FastMap, (6) may result in a mapping that violates the contractive property. A counterexample that shows this has four objects, a, b, c, and e, with distances $d(a,b) = 10$, $d(a,c) = 4$, $d(a,e) = 5$, $d(b,c) = 8$, $d(b,e) = 7$, and $d(c,e) = 1$. Here, a and b are pivots in the first iterations, resulting in $x_e - x_c = 6/5 = 1.2 > 1 = d(c,e)$. On the other hand, (8), can indirectly cause a violation of the contractive property in the next iteration of FastMap since d_H may not satisfy the triangle inequality (thereby failing to be a distance metric). A counterexample that demonstrates such a d_H also has four objects, a, b, c, and e, with distances $d(a,b) = d(c,e) = 6$, $d(a,c) = 5$, $d(a,e) = d(b,e) = 4$, and $d(b,c) = 3$. Given a and b as pivots can be found to yield $d_H(a,c) + d_H(a,e) \approx 5.141 < 5.850 \approx d_H(c,e)$, which violates triangle inequality. Finally, it is possible to show that, if the triangle inequality is violated for any of the distances between the two pivot objects r and s and an arbitrary object a, then the contractive property is violated for $d(r,a)$, $d(s,a)$, or both. (See [24] for detailed proofs on the claims in this paragraph.)

Observe that, when the mapping F_i produced by FastMap in i iterations is noncontractive, the value of $d_{i+1}(a,b)^2$ in (10) is negative for some $a, b \in S$. Thus, a and b cannot be used as a pivot pair in iteration $i+1$. If $d_{i+1}(a,b)^2 \leq 0$ for sufficiently many objects, the pivot-finding process may need more than a constant number of iterations to locate a pair with a positive negative distance. Furthermore, the fact that d_i may not be a distance metric for $i \geq 2$ (i.e., as pointed out above about $d_H = d_2$) means that the lower bound derived in Section 4.3 on the distance between the pivot objects need not hold.

5.3 Large Distortion with FastMap

As stated in the previous section, the embedding F produced by FastMap may fail to satisfy the contractive property. In this section, we are concerned with how much larger the distances in the embedding space can be compared to the original distances. We call this the *expansion* of F, defined as

$$\max_{o_1, o_2 \in \mathbb{U}} \left\{ \frac{\delta(F(o_1), F(o_2))}{d(o_1, o_2)} \right\}.$$

Note that, if the expansion is no greater than 1, F is contractive with respect to $\mathbb{U}$. Furthermore, if we can derive an upper bound c on the expansion, then any embedding F produced by FastMap can be made to be contractive by defining $\delta(o_1, o_2) = d_E(F(o_1), F(o_2))/c$ such that the expansion with respect to this δ is no more than 1. Unfortunately, the expansion of embeddings produced by FastMap when determining even just one coordinate is already relatively large, and for additional coordinates the expansion can be very large, especially if the heuristic discussed in Section 4.7 is not employed. These results are only sketched below, while a more detailed exposition can be found in [24].

5.3.1 Original Formulation

With the original formulation of FastMap (i.e., without the non-Euclidean heuristic), it is possible to derive the tight upper bound of 3 on the expansion for determining just one coordinate (i.e., $k = 1$) [24]. The proof of the tightness of the upper bound uses arbitrarily small distance values, which is clearly unrealistic. Nevertheless, even with a ratio of the largest distance value to the smallest of only 10, the expansion can be as large as 2.6.

When determining more than one coordinate with FastMap, we may obtain an arbitrarily large expansion, due to the fact that the triangle inequality does not necessarily hold for the distance functions used in the second and subsequent iterations of FastMap. The basic problem is that for iteration $i > 1$ the value of $d_i(r,s)^2$, as defined in (10), can be arbitrarily close to zero. One solution is to set a strict lower bound on the distance between the pivot objects, such that, if r and s are the first two pivot objects, then any pivot objects t and u chosen in any subsequent iteration i must obey $d_i(t,u) \geq d(r,s)/\beta$, where $\beta > 1$ is some constant. Unfortunately, this requirement means that the pivot-finding heuristic may no longer succeed in finding a legal pair of pivots in a constant number of iterations, thus possibly leading to $O(N^2)$ distance computations. An alternative solution is to terminate FastMap if a legal pivot pair is not found in $O(1)$ iterations of the pivot-finding process (see Section 4.3). This means that we may obtain fewer coordinate axes than desired. However, this is usually not a problem, as a low number of coordinate axes is preferable in most applications. Nevertheless, it is still not clear that the original distances are adequately approximated in the embedding space in cases when legal pivots cannot be found in $O(N)$ distance computations. This is a subject for further study.

Based on the distance range limitation $d_i(t,u) \geq d(r,s)/\beta$ ($\beta > 1$) on the pivots t, u used in iteration i, it is possible to show that the expansion is no more than $36\beta^2$ for $i = 2$ (i.e., after two iterations) and, in general, $O(\beta^{2(i-1)})$; see [24]. Since these bounds are not tight, we constructed nonlinear optimization models to obtain examples that yield a high expansion. For $i = 2$, the maximum expansion resulting from the model appeared to be at least proportional to β. For example, for $\beta = 5$ (which meant that the largest distance value was never more than 10 times larger than the smallest

one), the optimizer was able to discover a legal assignment of distance values that yielded an expansion of about 30. For $\beta = 50$, it yielded an expansion of about 300, and for $\beta = 100$, it yielded an expansion of about 600. Similarly, for $i = 3$, we obtained suggestive evidence that the upper bound is $O(\beta^{k-1})$. Admittedly, the largest values for the expansion result from a large number of significant digits (since this allows for very small relative differences in distance values). Still, with only five significant digits for the distance values, the expansion can be as high as several hundred.

5.3.2 Non-Euclidean Heuristic

When employing the heuristic described in Section 4.7, the upper bound on the expansion is still three, since the heuristic does not affect the first iteration of FastMap. However, the expansion due to the second and subsequent coordinate values will typically be less when the heuristic is used, as discussed in Section 4.7. The reason is that the worst-case expansion due to an iteration of FastMap is roughly proportional to the magnitude of the (squared) distance values in (9). For two coordinate values (i.e., $i = 2$), it is possible to derive an upper bound of $32\beta^2$, or nearly as much as when not using the heuristic (although, as before, this upper bound is not attainable). Modifying the nonlinear optimization model to account for the heuristic, we obtained an expansion of up to seven for two iterations of FastMap, and 14 for three iterations. Unfortunately, the optimizer appeared to have difficulty converging on solutions for the complicated formulas that result, so it is possible that it missed solutions with larger expansion values. Nevertheless, even though these values are much smaller that we saw before, they constitute a significant distortion in distance values.

When the heuristic is used, it is actually possible to use r and s as the pivot pair even when $d_i(r, s)$ is negative. Therefore, any pair of objects can be used as pivots, which is not the case without the heuristic (i.e., when $d_i(r, s)^2$ is negative). However, the effect on the quality of the mapping when $d_i(r, s)$ is negative for the pivot objects r and s is not obvious, so it is not clear whether it is advisable to allow such pivot objects. Furthermore, if $|d_i(r, s)|$ is very small, the expansion can potentially get very large. Thus, in order to keep the expansion from becoming too large, we must still be careful about how the pair of pivot objects is chosen.

5.3.3 Summary

A very large expansion, as we have seen is possible with FastMap, is problematic in similarity search (potentially causing a large number of false dismissals), and also in other applications such as clustering. In particular, FastMap may cause two nearby objects to appear to be far apart, resulting in these objects not being clustered together as they should be. Of course, it could be argued that expansion as we have defined it is a worst-case quality measure over all pairs of objects, and that the large upper bounds on expansion are not often attained. Nevertheless, determining the expected expansion for some distribution of distance values is difficult and, thus, remains an open question. Perhaps a more practical approach to the evaluation of the expansion in FastMap is to apply it to a suite of realistic data sets and measure the average expansion. Such a study is left for future work.

6 MetricMap

Wang et al. [43], [44] propose an interesting embedding, termed *MetricMap*, which is closely related to FastMap and SVD. In particular, like FastMap, MetricMap is based on an analogy to rotation and projection in Euclidean spaces. However, MetricMap differs from FastMap in that the embedding space is *pseudo-Euclidean*, which means that some coordinate axes make a negative contribution to "distances" between the points. Like FastMap, MetricMap uses $2k$ "pivot objects" (termed reference objects in MetricMap) in deriving the embedding function when mapping into a space of dimension k. However, mapping each object is less expensive in MetricMap than in FastMap, in that only $k + 1$ distance computations are necessary. Furthermore, MetricMap employs a different strategy to handle non-Euclidean metrics than FastMap (see Section 4.7), namely, by mapping into a pseudo-Euclidean space, which may result in less distortion in the distances. Nevertheless, as we show in Section 7.1, the result of MetricMap, like that of FastMap, may not be contractive.

Below, we first present a concise definition of MetricMap (Section 6.1), where we also informally describe the underlying intuition, though without going into details. Next, we show how SVD can be applied to yield a dimension reduction mapping in Euclidean spaces (Section 6.2), and argue that this is in fact equivalent to the formulation of MetricMap [24]. This equivalence helps to explain the underlying intuition behind the formulation of MetricMap, which may be difficult to grasp from the treatment of the method by Wang et al. [43], [44]. Section 6.3 describes the intuition behind a dimension-reduction mapping that is a part of MetricMap, and presents one method for achieving it.

6.1 Definition

Given a finite metric space (S, d), $S \subset \mathbb{U}$, MetricMap results in an embedding F into the space $(\mathbb{R}^k, \delta)$, where, as we shall see, δ is based on a pseudo-Euclidean squared distance function and, thus, is not necessarily a metric. The embedding is produced based on a subset $S' \subset S$ of size $m + 1$, $S' = \{o_0, o_1, \ldots, o_m\}$, where $m \geq k$ ($m = 2k$ is suggested by Wang et al. [43]). The set S' is used to form an imaginary space based on the "vectors" $(o_0, o_1), (o_0, o_2) \ldots (o_0, o_m)$, from which an "orthonormal basis" in $\mathbb{R}^k$ is derived that (approximately) "spans" the same space. Informally, the embedding $F(o)$ of an arbitrary $o \in \mathbb{U}$ can then be considered as involving "translating" o based on o_0 and "projecting" the result onto the k basis vectors. In this section, we present the mechanics of MetricMap, but without explaining why the above informal description applies to the method; its basis is the fact that, as argued in Section 6.2 (and shown in [24]), MetricMap is equivalent to applying translation and projection when S is drawn from Euclidean space.

Define the $m \times m$ matrix M based on elements of S', where

$$m_{i,j} = \frac{d(o_0, o_i)^2 + d(o_0, o_j)^2 - d(o_i, o_j)^2}{2}. \quad (12)$$

Since M is symmetric, the decomposition $M = QDQ^T$ exists, where $D = \mathrm{diag}(\lambda_i)$ is an $m \times m$ diagonal matrix containing the eigenvalues λ_i of M along the diagonal and Q is an $m \times m$ orthogonal matrix (i.e., $Q^{-1} = Q^T$) with the corresponding eigenvectors as columns. Furthermore, assume that the eigenvalues are ordered by decreasing

absolute value, i.e., such that $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_m| \geq 0$. Since some of the eigenvalues may be negative or zero, we define the $m \times m$ matrices C and J:

$$C = \text{diag}(|\lambda_i|_0), \text{ and} \quad (13)$$

$$J = \text{diag}(\text{sign}(\lambda_i)), \quad (14)$$

where

$$|\lambda_i|_0 = \begin{cases} |\lambda_i| & \text{if } \lambda_i \neq 0, \text{ and} \\ 1 & \text{if} \lambda_i = 0 \end{cases}$$

$$\text{sign}(\lambda_i) = \begin{cases} 1 & \text{if } \lambda_i > 0, \\ 0 & \text{if } \lambda_i = 0, \text{ and} \\ -1 & \text{if } \lambda_i < 0. \end{cases}$$

Observe that $D = C^{1/2} J C^{1/2}$ and that $C^{1/2} = \text{diag}(\sqrt{|\lambda_i|_0})$.

As defined by Wang et al. [43]), the set S' used to derive M is of size $2k+1$ (i.e., $m = 2k$). However, the final embedding is based on a set $R \subset S'$ of $k+1$ *reference objects*. In addition to o_0, R includes those objects o_i for which the set of "vectors" (o_0, o_i) most nearly "spans" the imaginary space formed based on S' (see Section 6.3 for details). Without loss of generality, we assume that $R = \{o_0, o_1, \ldots, o_k\}$; of course, R equals S' if $k = m$.

Now, for $l \leq m$, define the function $H_l : \mathbb{W} \rightarrow \mathbb{R}^l$,

$$H_l(o) = \left(\frac{d(o_0, o)^2 + d(o_0, o_j)^2 - d(o, o_j)^2}{2} \right)_{j \in \{1,\ldots,l\}}, \quad (15)$$

and the function $F_l : \mathbb{W} \rightarrow \mathbb{R}^l$,

$$F_l(o) = C_{[l,l]}^{-1/2} Q_{[l,l]}^{-1} H_l(o), \quad (16)$$

where the notation $A_{[i,j]}$ denotes the $i \times j$ principal submatrix of A. The embedding produced by MetricMap is $F = F_k$. (The form of the mapping presented in [43] has a redundant $J_{[k,k]}$ factor.) Clearly, in mapping a given object o by computing $F(o)$, it is necessary to perform $k+1$ distance computations $d(o, o_i)$ for $o_i \in R$; while to form the embedding function F (i.e., when computing the matrix M) $O(m^2) = O(k^2)$ distance computations are needed.

For $l \leq m$, define the function $\Delta_l : \mathbb{R}^l \times \mathbb{R}^l \rightarrow \mathbb{R}$:

$$\Delta_l(x, y) = (x - y)^T J_{[l,l]} (x - y) = \sum_{i=1}^{l} \text{sign}(\lambda_i)(x_i - y_i)^2. \quad (17)$$

Wang et al. [43] suggest defining the "distance" function $\delta : \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}$ on the transformed space as follows:

$$\delta(x, y) = \begin{cases} \sqrt{\Delta_k(x, y)} & \text{if } \Delta_k(x, y) \geq 0, \\ -\sqrt{-\Delta_k(x, y)} & \text{otherwise.} \end{cases}$$

The Δ_k function, used in defining δ, represents a pseudo-Euclidean "square distance" function based on the pseudo-Euclidean "inner product" $\langle x, y \rangle = x^T J_{[k,k]} y = \sum_i s_i x_i y_i$, where $s_i \in \{-1, 0, 1\}$. Strictly speaking, if $s_i \leq 0$ for some $i \in \{1, \ldots, k\}$, then $\langle \cdot, \cdot \rangle$ is not an inner product as $\langle x, x \rangle$ can be negative and $\langle x, x \rangle = 0$ need not imply that x is the zero vector. Notice that $\langle \cdot, \cdot \rangle$ resembles the dot product of Euclidean spaces, $x \cdot y = x^T y$, from which the Euclidean norm and the Euclidean distance metric are derived. However, δ is not a distance metric if any of the eigenvalues λ_i are zero (since, then, $\delta(x, y) = 0$ does not necessarily imply $x = y$), and if any of the eignevalues are negative, then δ cannot be properly said to be a distance function (since, then, $\delta(x, y) < 0$ for some $x, y \in \mathbb{R}^k$).

Observe that coordinate axes that correspond to zero eigenvalues do not contribute to distances according to δ (since the corresponding value on the diagonal of J is then zero). Thus, if only $k' < k$ eigenvalues are nonzero, we can actually map into $\mathbb{R}^{k'}$ instead of $\mathbb{R}^k$ by using a reference set R of only $k'+1$ objects and using $F_{k'}$ as the final embedding (alternatively, only the first k' coordinate values in F_k could be used). However, as we point out in Section 6.3, the hope is that by constructing a matrix M that is larger than $k \times k$ (i.e., based on a set S' that is larger than $k+1$), we obtain at least k nonzero eigenvalues.

6.2 Motivation: SVD

In the special case that S is drawn from a Euclidean space, it is possible to show that MetricMap is equivalent to applying SVD for dimensionality reduction. In particular, assume that $S \subset \mathbb{W} = \mathbb{R}^n$ and that d is the Euclidean metric d_E (i.e., $d(a, b) = \|a - b\|_2$, where $\| \cdot \|_2$ denotes the Euclidean norm). Furthermore, as before, let S' be a subset of S of size $m+1$; here, however, we denote the elements of S' with p_i. Now, based on S', we define the $n \times m$ matrix P, where $P = [\, s(p_1)\ (p_2)\ \ldots\ s(p_m)\,]$ and $s(p_i) = p_i - p_0$. In other words, the column vectors in P equal the m vectors $\overrightarrow{p_0 p_i}$ for $p_i \in S' \setminus \{p_0\}$. The singular value decomposition (SVD) of P has the form $P = U \Sigma V^T$, where $U = [u_1\ u_2\ \ldots\ u_n]$ and $V = [\, v_1\ v_2\ \ldots\ v_m\,]$ are $n \times n$ and $m \times m$ orthogonal matrices, respectively, and $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_m)$ is an $n \times m$ matrix and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$, termed the *singular values* of P (at least $m - n$ of the singular values are zero if $m > n$).

Below, we show that for Euclidean spaces, there is an intimate connection between the matrix U and the MetricMap embedding as presented in (16). Before we do so, however, we review some properties of the matrices in the SVD of P. First, observe that the bottom $n - m$ rows of Σ contain only zeros, i.e.,

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_m \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

For notational convenience, let Σ_l be the $l \times m$ principal submatrix of Σ, i.e., the first l rows, and let $U_l = [\, u_1\ u_2\ \ldots\ u_l\,]$, i.e., the first l columns of U. Note that Σ_m is a diagonal matrix, and that $U\Sigma = (\sigma_i u_i)_{i \in \{1,\ldots,m\}} = U_m \Sigma_m$. U and V being orthogonal means that $U^T U = U U^T = I_n$ and $V^T V = V V^T = I_m$, where I_l denotes the $l \times l$ identity matrix. Multiplying a vector by an orthogonal matrix establishes new coordinates for the vector, amounting to a rotation and, thus, preserves its length under the Euclidean norm. For example, if $p \in \mathbb{R}^n$, then $(u_i \cdot p) u_i$ is an orthogonal projection of p onto u_i since $\|u_i\|_2 = 1$, so $U^T p$ represents the coordinates of p using the column vectors of U as a basis, and

$$\|U^T p\|_2^2 = (U^T p) \cdot (U^T p) = (U^T p)^T (U^T p)$$
$$= p^T U U^T p = p^T I_m p = p^T p = p \cdot p = \|p\|_2^2.$$

Since $\|p\|_2 \geq 0$, this implies that $\|U^T p\|_2 = \|p\|_2$.

Now, for $l \leq n$, define the mapping $f_l : \mathbb{R}^n \rightarrow \mathbb{R}^l$ such that

$$f_l(p) = U_l^T (p - p_0) = (u_i \cdot (p - p_0))_{i \in \{1,\ldots,l\}}. \quad (18)$$

Thus, $f_n(p)$ amounts to a translation followed by a rotation (since $U_n = U$) and, therefore, preserves the Euclidean distances between any two points in $\mathbb{R}^n$ since Euclidean distances are invariant under rotation:

$$\|f_n(p_i) - f_n(p_j)\|_2 = \|U^T(p_i - p_0) - U^T(p_j - p_0)\|_2$$
$$= \|U^T(p_i - p_j)\|_2 = \|p_i - p_j\|_2,$$

where $p_i, p_j \in \mathbb{R}^n$. Furthermore, observe that $f_l(p)$ consists of the first l coordinate values in $f_n(p)$ and represents a translation followed by a projection on the first l orthonormal column vectors in U.

The connection between SVD and MetricMap can be seen from the fact that the functions F_m according to (16) and f_m according to (18) are, in fact, equivalent. In other words, for $p \in \mathbb{R}^n$, $F_m(p) = f_m(p)$ [24].

At this point, it is instructive to verify that the embedding f_m according to (18) satisfies the informal description of MetricMap presented at the beginning of Section 6.1, assuming that $k = m$. First, observe that f_m is the result of translating by p_0 (i.e., o_0) and projecting onto a set of the m orthonormal vectors in U_m. Thus, assuming that the orthonormal basis U_m spans the proper space, f_m satisfies the description of F in Section 6.1. Second, the space spanned by $\overrightarrow{p_0 p_i}$ (i.e., (o_0, o_i) in the notation of Section 6.1), the column vectors in P, is in fact the same as that spanned by U_m, completing the picture. To see why, consider the $n \times m$ matrix $U^T P = \Sigma V^T$, the result of projecting each column vector in P onto the orthonormal basis U. Since the last $n - m$ rows of Σ contain only zeros, the last rows of ΣV^T also contain only zeros since

$$\Sigma V^T = \begin{bmatrix} \Sigma_m \\ Z \end{bmatrix} V^T = \begin{bmatrix} \Sigma_m V^T \\ Z V^T \end{bmatrix} = \begin{bmatrix} \Sigma_m V^T \\ Z \end{bmatrix},$$

where Z is the $(n - m) \times m$ zero matrix. Thus, in the coordinate system defined by U, the last $n - m$ coordinate values of the vectors $\overrightarrow{p_0 p_i}$ are zero, so the vectors span the same space as the vectors in U_m. (More precisely, if P has any zero singular values and $\sigma_{m'+1}$ is the first zero singular value, P spans the same space as $U_{m'}$.) Furthermore, this implies that f_m actually preserves the distances among the objects in S' (it is possible to show that F_m has the same property, regardless of the metric space).

The function f_m according to (18) comprises a dimensionality reduction from $\mathbb{R}^n$ to $\mathbb{R}^m$, assuming that $m < n$. The customary manner in which the dimensionality is reduced using SVD is to use the $n \times N$ matrix $P = (s(p))_{p \in S}$ as the basis, where $s(p) = p - \mu$ and μ is the center of gravity of S (i.e., each coordinate value of μ is the mean over all coordinate values of the points in S along the same axis). Then, given U in the SVD of P, the function $g_m(p) = U_m^T p$ reduces the dimensionality of points in S from n to m in a manner that results in the least-square error. However, if N is large, computing the dimension reduction mapping based on the entire set S is expensive, so basing the dimension reduction mapping on a subset of S, as done by MetricMap, may be worthwhile.

6.3 Reducing Dimensionality from m to k

As we saw in Section 6.1, the embedding $F = F_k$ produced by MetricMap ((16)) is derived based on the $m = 2k$ "vectors" (o_0, o_i), but the embedding itself maps into $\mathbb{R}^k$. The reasoning behind using more "vectors" m than the final dimensionality k is that this can be expected to provide an embedding that better approximates distances in the original space,[6] while performing the same number of distance computations (i.e., $k + 1$) in computing $F(o)$ for $o \in \mathbb{U}$.

A good way to get an intuition for this is to examine the case when S is drawn from a Euclidean space; similar arguments can be made when this is not the case. In particular, observe that the extent to which the mapping f_m, according to (18), preserves distances of the points in S depends on the extent to which the distribution of the points in S' approaches the distribution of the points in S. If S' is a random sample of S, we can therefore expect the preservation of distances to improve as S' is enlarged. Furthermore, there is a higher probability that the column vectors $\overrightarrow{p_0 p_i}$ of P span a space of dimensionality k (i.e., implying that $\sigma_k > 0$) if P has more than k column vectors, i.e., if S' is larger than $k + 1$. Moreover, since the variance along coordinate axis i in $f_m(p)$ for $p \in S'$ is proportional to σ_i (more precisely, equals σ_i^2) and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$, the expected contribution of axis i to $\delta(f_m(p), f_m(p'))$, where $p, p' \in S'$, is greater than that of axis $i + 1$. Thus, the best distance preservation for the points in S' given only k out of the m coordinate values produced by f_m is obtained with the first k coordinate values, i.e., based on f_k, so using f_k is a reasonable heuristic for obtaining good distance preservation for the points in S.

Incidentally, F_k is actually only an approximation of f_k when S is drawn from a Euclidean space. In particular, $F_k(p) = C_{[k,k]}^{-1/2} Q_{[k,k]}^{-1} H_k(o)$, while

$$f_k(p) = U_k^T (p - p_0) = C_{[k,k]}^{-1/2} Q_{[m,k]}^T H_m(o)$$

(this equality is obtained from $U^T = \Sigma^{-1} V^T P^T$, which holds when all singular values are nonzero). In other words, F_k is based on H_k while f_k is based on H_m. A similar argument can be made about F_k when S is not drawn from a Euclidean space. However, given $m = 2k$, $H_m(p)$ is twice as expensive to compute as $H_k(p)$, so basing F_k on the latter trades off accuracy for time. Nevertheless, since f_k already involves a heuristic (in that it is based on a subset of S), this tradeoff may be worthwhile.[7]

Notice that, although perhaps a rare situation, it is possible for $Q_{[k,k]}$ to be noninvertible, in which case F_k cannot be evaluated. The originators of MetricMap do not mention this as a possibility and, thus, do not provide a solution. This problem does not occur if we use F_m since $Q_{[m,m]} = Q$ and $Q^{-1} = Q^T$. Thus, at worst, we can use the first k coordinate values in F_m as the final embedding (of course, if $m = k$, then $F_k = F_m$).

6. Of course, the implicit assumption is made that the distribution of distances in (S, d) is similar as in $(S \cup Q, d)$, where Q is the set of query objects.

7. However, when d is the Euclidean metric, it is better to use f_k rather than F_k since the former is computed without using the reference objects, i.e., based on U_k.

Wang et al. suggest that the set R of $k+1$ reference objects be obtained from the larger set S' by retaining o_0 and the k other objects that "span" $\mathbb{R}^k$. However, exactly how to do this was not described. There are several ways to proceed; Zhang [45] suggests the following method. First, sort the objects in $S' \setminus \{o_0\}$ in increasing order by $|d(o_i, o_0)^2 - \Delta_k(F'(o_i), F'(o_0))|$, where $F'(o)$ consists of the first k coordinate values in $F_m(o)$. In other words, the ordering is by how well the distance between o_i and o_0 is preserved by F'. Next, initialize R as containing only o_0. Choose each object $o_i \in S' \setminus \{o_0\}$ in turn in the above order. If $H_i(o_i)$ is close to being a linear combination of the mapping of the existing objects in R, then o_i is rejected[8] (to simplify the notation, we assume here that $R = \{o_0, o_1, \ldots, o_{i-1}\}$). Otherwise, add o_i to R. This process is terminated once $|R| = k+1$.

7 MetricMap: Properties and Improvements

Since MetricMap, like FastMap, is inspired by properties of Euclidean space, much of the discussion in Section 5 also applies to MetricMap. In particular, we show in Section 7.1 that the embedding produced by MetricMap is typically not contractive when d is not the Euclidean metric. Furthermore, even when d is the Euclidean metric, MetricMap may still not be contractive. Nevertheless, MetricMap can always achieve a distance-preserving embedding with respect to S, given a sufficient number of coordinate axes (i.e., $N+1$, and perhaps less). The remaining sections discuss other properties of MetricMap and propose minor variants. In Section 7.2, we point out that the fact that some coordinate axes have a negative contribution to distances can make similarity search expensive, and suggest an alternative definition of δ that does not result in negative values. In Section 7.3, we introduce a heuristic for choosing the "origin" of the embedding, designed to reduce the error in distances over S. In Section 7.4, we explore the relationship between FastMap and MetricMap in more detail, and show that FastMap can be modified to result in an embedding into pseudo-Euclidean space, like MetricMap.

7.1 Noncontractiveness

As mentioned in Section 6.2, when S is drawn from a Euclidean space, the mapping F_m according to (16) is equivalent to applying translation and rotation on the points in P and retaining m of the resulting coordinate values. Thus, since the Euclidean metric is invariant to translation and rotation, MetricMap results in a contractive embedding if $k = m$ (i.e., the whole set S' is used as reference objects). However, if $m > k$ and the mapping F_k is used instead (i.e., based on $k+1$ reference objects instead of $m+1$, chosen as described in Section 6.3), the embedding is not necessarily contractive. For example, suppose that $k = 1$ and that S' consists of the $2k+1 = 3$ points $p_0 = (0,0)$, $p_1 = (-2,1)$, and $p_2 = (0,2)$. In this case, the procedure described in Section 6.3 would obtain $R = \{p_0, p_1\}$, and the result of the embedding would be $F_1(p_0) = 0$, $F_1(p_1) \approx 2.5$, and $F_1(p_2) \approx 1.0$. Thus, the distance in the embedding space between $F_1(p_0)$ and $F_1(p_1)$ would be greater than that between p_0 and p_1 since $d(p_0, p_1) = \sqrt{2^2 + 1^2} \approx 2.2$, while $\delta(F_1(p_0), F_1(p_1)) \approx 2.5$.

When (S, d) is an arbitrary finite metric space (i.e., not necessarily isomorphic to a subset of a Euclidean space), it can be shown that F_m preserves the distances among the objects in S'. Unfortunately, this is not enough to guarantee that F_m is contractive, and, indeed, F_m will typically not be contractive. The source of the noncontractiveness is essentially the same as that for FastMap, as detailed in Section 5.2. In particular, as in FastMap, the mapping F_m is based on applying an analogy to translation and rotation, but when d is not the Euclidean metric, the analogy to rotation is not guaranteed to preserve distances.

For a single coordinate axis, MetricMap can result in the same amount of expansion as FastMap, i.e., at most three as derived in Section 5.3. The expansion for more coordinate axes can be even greater, as for FastMap. Nevertheless, since negative eigenvalues have a negative contribution to "distances" according to δ, it is possible that the excessive distortion exhibited by FastMap is ameliorated, at least to some extent.

7.2 Distances in the Embedding Space

The "distance" function δ for MetricMap is based on the function Δ_k, as defined by (17). Thus, coordinate axes corresponding to negative eigenvalues have a negative contribution to the distance values according to δ (since J is defined based on the signs of the eigenvalues). There may be a major problem with such distance functions for similarity search applications where a spatial index is used to organize the result of the embedding, in that the query region in the mapped space may be very large, even for a range query with a small radius. For example, suppose that we have a two-dimensional pseudo-Euclidean space with a positive x-axis and negative y-axis, i.e., $\lambda_1 > 0$ and $\lambda_2 < 0$. Given a query object $q \in \mathbb{U}$, mapped into $q' = F(q) = (x_q, y_q)$, consider the two diagonal lines through (x_q, y_q), i.e., the lines given by the equations $x - x_q = y - y_q$ and $x - x_q = -(y - y_q)$ which are at 45 and -45 degree angles from the x-axis. The two lines can be viewed as dividing the plane into four quadrants; we abbreviate them with N, E, S, and W, in clockwise order, where the N quadrant is the one "above" (x_q, y_q). For any point $p \in \mathbb{R}^2$ on these lines, $\delta(q', p)$ is zero, while if p is in the N or S quadrants (and not on the lines), $\delta(q', p)$ is negative. Thus, for a range query with radius ϵ, the query region in the embedding space, defined by $\delta(q', p) \leq \epsilon$, would include the entire N and S quadrants, in addition to portions of the W and E quadrants.[9] Hence, since the N and S quadrants are already infinite, the query region is also infinite (although, in practice, it would only reach to the boundary of the data space). Therefore, even a search region with a very small radius may contain nearly all the objects, and perhaps overlap all the leaf node regions of the index. A possible solution to this dilemma is to retain only dimensions corresponding to positive eigenvalues, thus embedding into a Euclidean space, but this may lead to excessive expansion (see Section 5.3).

8. This heuristic is based on the fact that, if a vector v_i is a linear combination of a set of vectors $v_1, \ldots, v_{i-1}$, then adding v_i to the set does not increase the span of the set of vectors. Also, note that, if $H_i(o_i)$ is close to being a linear combination of $H_i(o_j)$, $j < i$, then the $i \times i$ matrix with column vectors $H_i(o_j)$, $j \leq i$ has a determinant that is close to zero.

9. More precisely, the query region would be delimited by the two curves of a hyperbola, which lie in the W and E quadrants, respectively.

A related issue is the fact that δ can return negative distance values, which has a questionable purpose. In particular, when d returns a distance value of zero, complete similarity is implied, i.e., $d(a,b)=0$ means that $a=b$ (if d is a pseudometric, a distance of zero may not imply equality, but it at least implies close similarity or even equivalence). On the other hand, we cannot for example conclude that a and b are more similar when $\delta(F(a),F(b))$ equals -10 than when it equals -5. Thus, we propose the following alternative definition of δ that simply "truncates" the distances at zero:

$$\delta(x,y)=\begin{cases}\sqrt{\Delta_k(x,y)} & \text{if } \Delta_k(x,y)\geq 0,\\ 0 & \text{otherwise.}\end{cases}$$

7.3 Choosing the Origin

Consider the case when $S\subset\mathbb{U}=\mathbb{R}^n$ and d is the Euclidean metric. A dimension-reduction mapping $g_k:\mathbb{R}^n$ to $\mathbb{R}^k$, for a fixed $k<n$, is said to be result in the least-square error over a data set S if

$$\sum_{p_1,p_2\in S}\left(d(p_1,p_2)-\delta(g_k(p_1),g_k(p_2))\right)^2$$

is minimized. As mentioned in Section 4.1, it can be shown that a linear transformation with the least square error is one obtained by translating the points in S such that their center of gravity is at the origin, rotating such that the variance along the first m coordinate axes is as great as possible and then projecting onto those k coordinate axes. In other words, the function $g_k(p)=U_k(p-\mu)$ satisfies the property, where μ is the center of gravity, $P=(p-\mu)_{p\in S}$, and $P=U\Sigma V^T$ is the singular value decomposition of P.

Comparing the function g_k derived above with f_k according to (18), it should be clear that the closer p_0 is to the center of gravity for S', the smaller is the "square error" of f_k over S' is, the closer p_0 is to the center of gravity for S'. Moreover, the error of f_k over S should also tend to be lower for such a choice of p_0, assuming that the distribution of points in S' is similar as that of S (and, in particular, their centers of gravity are similar), which is ideally the case. In fact, rather than picking a random point in S as p_0, we may wish to construct $S'=\{p_0\}\cup S''$ such that $S''\subset S$ and p_0 is the center of gravity of the objects in $S''=\{p_1,\ldots,p_k\}$ (i.e., $p_0=\frac{\sum_i p_i}{k}$). We may even wish to set p_0 to μ, the center of gravity for all of S. If the whole data set S fits in main memory, computing its center of gravity does not add significant overhead. However, if S is large enough to exceed available memory, then computing the center of gravity requires an extra scan of the entire data set on disk, thus potentially performing nearly twice the amount of I/O for embedding S.

When d is not the Euclidean metric, we do not have the luxury of computing the center of gravity directly. Nevertheless, a heuristic for choosing o_0, the "origin" of the embedding, from among the objects in S' can be derived by examining the Euclidean case. In particular, it can be shown [24] that the sum of the squared distances between p_0 and the remaining objects in S' depends on the squared distance between p_0 and μ', the center of gravity for S'. Thus, since $\sum_i\|p_i-\mu'\|_2^2$ is independent of p_0, the sum $\sum_i\|p_i-p_0\|_2^2$ of squared distances increases as p_0 is farther away from μ', the center of gravity for S'. Therefore, the best choice of p_0 from among the objects in S' is the one that minimizes the sum of squared distances since this indicates that p_0 is close to μ'. It should be clear that this strategy can be used as a heuristic when d is not the Euclidean metric. In other words, after choosing the subset $S'\subset S$, we compute the sum of squared distances for objects in S', and pick o_0 as the one for which the sum is minimized.

7.4 Relationship between FastMap and MetricMap

The formulas used in FastMap and MetricMap appear very similar, namely, (9) for FastMap and (15) for MetricMap. This is no accident, as (9) is based on an analogy to an orthogonal projection in Euclidean spaces, while (15) is based on an analogy to applying the dot product in Euclidean spaces and an orthogonal projection can be defined in terms of a dot product. In particular, if $p_1^1,p_2^1,o\in\mathbb{R}^m$ and d_1 is the Euclidean distance metric, then (9) for the first iteration of FastMap is equivalent to

$$x_o^1=u_1\cdot(o-p_1^1)=\frac{(p_2^1-p_1^1)\cdot(o-p_1^1)}{\|p_2^1-p_1^1\|_2},$$

where u_1 is the unit vector parallel to $p_2^1-p_1^1$. The vector $(u_1\cdot(o-p_1^1))u_i$ is the orthogonal projection of $p_2^1-p_1^1$ onto u_1. If U is an orthogonal matrix such that $U^Tu_1=v_1$, where $v_1=(1,0,\ldots,0)$, then u_1 is the first column in U. Thus, x_o^1 is the first component of $U^T(o-p_1^1)$, which clearly is of the same form as (18). Furthermore, as shown in [24], the latter equation is equivalent to (16), which forms the basis of the MetricMap embedding.

In light of the similarity between the equations for FastMap and MetricMap, it is tempting to think that they are equivalent when determining m coordinate axes if the pairs $(o_0,o_i),i\in\{1,\ldots,m\}$ were used in FastMap, i.e., $p_1^i=o_0$ and $p_2^i=o_i$ for $i\in\{1,,m\}$. In this case, the first component in $H_m(o)$ according to (15) would be $x_o^1d(p_2^1,p_1^1)$, so the first component in $F_m(o)$, according to (16), would be x_o^1 if we set $q_1=e_1$ and $\lambda_1=d(p_2^1,p_1^1)$. Thus, we may ask whether the mapping of o according to FastMap can be obtained by a suitable rotation of $H_m(o)$. Unfortunately, although true if d is the Euclidean metric, this answer is no for arbitrary metric spaces.

The notion of pseudo-Euclidean squared distances used by MetricMap can be adopted for use in FastMap, replacing the non-Euclidean heuristic described in Section 4.7. In particular, the definition of x_o^i in (9) would be replaced by

$$x_o^i=\frac{d_i(p_1^i,o)^2+d_i(p_1^i,p_2^i)^2-d_i(p_2^i,o)^2}{2|d_i(p_1^i,p_2^i)|},$$

and the definition of $d_i(a,b)^2$ in (10) would be replaced by

$$d_i(a,b)^2=d_{i-1}(a,b)^2-\operatorname{sign}\big(d_{i-1}(p_1^i,p_2^i)\big)\big(x_a^{i-1}-x_b^{i-1}\big)^2.$$

Furthermore, in the mapped space, we would use the pseudo-Euclidean squared distance function Δ, where

$$\Delta(x,y)=\sum_{i=0}^{k}\operatorname{sign}\Big(d_{i-1}(p_1^i,p_2^i)^2\Big)(x-y)^2.$$

TABLE 1
Summary of Embedding Methods

	SparseMap (original/proposed)	FastMap	MetricMap
Embedding Space	Euclidean/L_∞	Euclidean	Pseudo-Euclidean
Construction Cost	$O(kN)$	$3kN$	$4k^2-(N-2k)(k+1)$
Mapping Cost	$O(k)$	$2k$	$k+1$
Distortion	$\sqrt{2}$/None	High?	High?
Contractive	No/Yes	Only for Euclidean	Only for Euclidean

Note that with these definitions, (p_1^i, p_2^i) is still a legal pivot pair even if $d_i(p_1^i, p_2^i)^2$ is negative. Furthermore, $d_{i+1}(p_1^i, p_2^i) = 0$ even in this case, whereas this would not be true with the heuristic described in Section 4.7 for such pivot objects (although $d_{i+1}(p_1^i, p_2^i) > 0$). Using the above pseudo-Euclidean heuristic in FastMap has the same potential drawbacks as using dimensions corresponding to negative eigenvalues in MetricMap. Moreover, for the purposes of choosing the pivot pair in iteration i, it is not clear whether it is better to pick the pair (p_1^i, p_2^i) such that $d_{i-1}(p_1^i, p_2^i)^2$ or $|d_{i-1}(p_1^i, p_2^i)^2|$ is as large as possible. Nevertheless, the above heuristic may be a competitive alternative to the heuristic in Section 4.7.

8 Concluding Remarks

We have evaluated a number of embeddings of finite metric spaces in the context of their usage for similarity searching with 100 percent recall. One hundred percent recall is important in similarity search as it ensures that no relevant object is dropped from the query response. Our focus was on the SparseMap, FastMap, and MetricMap embedding methods; a somewhat expanded treatment can be found in [24], including a discussion of Multidimensional Scaling. One hundred percent recall is achieved when the resulting embedding is contractive. Although Linial et al. [1] showed how to make the Lipschitz embeddings contractive, we showed that the speedup heuristics that comprise the SparseMap adaptation make the resulting mapping noncontractive. Moreover, we demonstrated how to modify the SparseMap heuristics so that the resulting embedding is indeed contractive.

Table 1 summarizes the properties of the three embedding methods and includes our proposed modification of SparseMap. In the table, construction cost refers to the cost of constructing the embedding function F, while mapping cost refers to the cost of computing $F(o)$ for some $o \in \mathbb{U}$, where the cost is in terms of the number of distance computations. In the cost formulas, k is the dimensionality of the embedding space, N the size of the data base, and we assumed that only two iterations are performed in FastMap when determining pivot objects (see Section 4.3). For SparseMap, we only list asymptotic costs, which are based on a suitable choice of embedding space dimensionality and heuristic parameters [21].

In the case of FastMap, we have first proven that, it is contractive when the data is drawn from a Euclidean space, and the distance metric in the embedding space is a Minkowski metric L_p with $p \geq 2$ (which includes the Euclidean distance metric). In their development of FastMap, Faloutsos and Lin [20] claimed that the advantage of FastMap over methods such as KLT is that FastMap can work for data drawn from an arbitrary metric space (i.e., the only information about the data objects consists of the interobject distances, which are required to satisfy the triangle inequality). However, we showed that FastMap is only a heuristic when the data is drawn from a metric space that is not Euclidean. In particular, we have proven that, in such a case, it is possible for FastMap not to be contractive. We showed that this was a direct result of the implicit assumption by Faloutsos and Lin [20] of the applicability of the Pythagorean theorem, which in the case of a general metric space can only be used as a heuristic in computing the projected distance values. In fact, this led to definitions of distance functions at intermediate iterations that did not satisfy the triangle inequality and thereby failed to be distance metrics. Noncontractiveness enabled us to prove the following properties of FastMap for this situation:

1. Given a value k, application of FastMap may not always be possible in the sense that we are not guaranteed to be able to determine k coordinate axes.
2. The distance distortion of the embedding can be very large, as evidenced by the bounds that we gave, some of which were attainable, on how much larger the distances in the embedding space can be.
3. The fact that we may not be able to determine k coordinate axes limits the extent of achievable distance preservation. However, more importantly, failure to determine more coordinate axes does not necessarily imply that relative distances among the objects are effectively preserved.
4. The presence of many nonpositive, or very small positive, distance values (which can cause large distortion) in the intermediate distance functions (i.e., those used to determine the second and subsequent coordinate axes) may cause FastMap to no longer satisfy the claimed $O(N)$ bound on the number of distance computations in each iteration. In particular, finding a legal pivot pair may, in the worst case, require examining the distances between a significant fraction of all possible pairs of objects, or $\Omega(N^2)$ distance computations.

A heuristic for handling non-Euclidean distance metrics in FastMap, proposed by Wang et al. [23], alleviates some of the drawbacks listed above. In particular, it should reduce the amount of distance distortion in the embedding, and the number of object pairs that do not qualify as pivots should be lower, thus reducing the likelihood of not satisfying the $O(N)$ bound on the number of distance computations in each iteration of FastMap. However, a detailed empirical study of

the effect of the heuristic on actual data sets remains to be performed.

MetricMap has many of the same properties as FastMap, given that both are inspired by dimension-reduction methods for Euclidean spaces. In particular, MetricMap is also contractive for data drawn from Euclidean spaces, but usually not for other types of data and metrics. However, when the number of "reference objects" is reduced in MetricMap, we showed that the embedding may be non-contractive even for data drawn from a Euclidean space (Section 7.1). Moreover, MetricMap can also lead to a large expansion. We point out that the fact that the MetricMap yields a pseudo-Euclidean embedding may lead to poor performance for similarity search when spatial indexes are employed for the mapped objects (Section 7.2). Furthermore, we proposed heuristics for choosing the origin of the embedding space, both for Euclidean spaces and for arbitrary metric spaces (Section 7.3). Like FastMap, MetricMap uses $2k$ "pivot objects" (termed reference objects in MetricMap) in deriving the embedding function when mapping into a space of dimension k. However, mapping each object is less expensive in MetricMap than in FastMap, in that only $k+1$ distance computations are necessary. Furthermore, MetricMap employs a different strategy to handle non-Euclidean metrics than FastMap (see Section 4.7), namely, by mapping into a pseudo-Euclidean space, which may result in less distortion in the distances.

Many possibilities exist for future work on general embedding methods. Our work, for example, suggests a number of possible experimental studies, some of which we have mentioned in the text. These include:

1. Compare the relative quality of the embedding resulting from the various methods on actual data sets.
2. Evaluate the added cost of the modified SparseMap heuristic that we suggested (Section 3.2), and the difference in the embedding quality compared to the original SparseMap heuristic on actual data sets.
3. Measure the distortion in distances that result when using FastMap on actual data sets, to see if it tends to approach the worst-case that we describe in Section 5.3, as well as study the extent to which the distortion affects nearest-neighbor search (since the distortion is greatest over small distances).
4. Study the effect of the heuristics that we propose for choosing the origin of the embedding space in MetricMap (Section 7.3).
5. Examine the difference in the quality of the embedding of using different heuristics for handling non-Euclidean distance metrics in FastMap, including the one proposed by Wang et al. [23] and the one that we suggest in Section 7.4.

Another interesting direction for future work is to study other general embedding methods (such as other variants of Lipschitz embeddings [34], [36]), and also evaluate them experimentally.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under Grants IRI-9712715, EIA-99-00268, EIA-99-01636, EAR-99-05844, and IIS-00-86162.

REFERENCES

[1] N. Linial, E. London, and Y. Rabinovich, "The Geometry of Graphs and Some of Its Algorithmic Applications," *Combinatorica*, vol. 15, pp. 215-245, 1995.

[2] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Reading, Mass.: Addison-Wesley, 1990.

[3] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading, Mass.: Addison-Wesley, 1990.

[4] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl, "3D Shape Histograms for Similarity Search and Classification in Spatial Databases," *Proc. Advances in Spatial Databases-Sixth Int'l Symp.*, R.H. Guting, D. Papadias, and F.H. Lochovsky, eds., pp. 207-226, July 1999.

[5] J.L. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, and W. Niblack, "Efficient Color Histogram Indexing for Quadratic Form Distance Functions," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 7, pp. 729-736, July 1995.

[6] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas, "Fast Nearest Neighbor Search in Medical Image Databases," *Proc. 22nd Int'l Conf. Very Large Data Bases*, T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, eds., pp. 215-226, Sept. 1996.

[7] R. Agrawal, C. Faloutsos, and A.N. Swami, "Efficient Similarity Search in Sequence Databases," *Proc. Fourth Int'l Conf. Foundations of Data Organization and Algorithms*, D.B. Lomet, ed., pp. 69-84, Oct. 1993.

[8] K.-P. Chan, A.W.-C. Fu, "Efficient Time Series Matching by Wavelets," *Proc. 15th IEEE Int'l Conf. Data Eng.*, pp. 126-133, Mar. 1999.

[9] H. Hotelling, "Analysis of a Complex of Statistical Variables into Principal Components," *J. Educational Psychology*, vol. 24, pp. 417-441, and pp. 498-520, 1933.

[10] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, second ed. Boston: Academic Press, 1990.

[11] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

[12] C.S. Burrus, R.A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer*. Upper Saddle River, N.J.: Prentice Hall, 1998.

[13] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*. Prentice Hall, 1995.

[14] D. Achlioptas, "Database-Friendly Random Projections," *Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 274-281, May 2001.

[15] E. Bingham and H. Mannila, "Random Projection in Dimensionality Reduction: Applications to Image and Text Data," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 245-250, Aug. 2001.

[16] N. Gershenfeld, *The Nature of Mathematical Modeling*. Cambridge, UK: Cambridge Univ. Press, 1999.

[17] J. Bourgain, "On Lipschitz Embedding of Finite Metric Spaces in Hilbert Space," *Israel J. Math.*, vol. 52, nos. 1-2, pp. 46-52, 1985.

[18] W. Johnson and J. Lindenstrauss, "Extensions of Lipschitz Mappings into a Hilbert Space," *Contemporary Math.*, vol. 26, pp. 189-206, 1984.

[19] N.J. Young, *An Introduction to Hilbert Space*. Cambridge, UK: Cambridge Univ. Press, 1988.

[20] C. Faloutsos and K. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," *Proc. ACM SIGMOD Conf.*, pp. 163-174, May 1995.

[21] G. Hristescu and M. Farach-Colton, "Cluster-Preserving Embedding of Proteins," technical report, Rutgers Univ., Piscataway, New Jersey, 1999.

[22] J.B. Kruskal and M. Wish, "Multidimensional Scaling," technical report, Sage Univ. Series, Beverly Hills, Calif., 1978.

[23] J.T.-L. Wang, X. Wang, K.-I. Lin, D. Shasha, B.A. Shapiro, and K. Zhang, "Evaluating a Class of Distance-Mapping Algorithms for Data Mining and Clustering," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 307-311, Aug. 1999.

[24] G.R. Hjaltason and H. Samet, "Contractive Embedding Methods for Similarity Searching in Metric Spaces," Computer Science TR-4102, Univ. of Maryland, College Park, Maryland, Feb. 2000.

[25] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions," *J. ACM*, vol. 45, no. 6, pp. 891-923, Nov. 1998.

[26] M. Bern, "Approximate Closest-Point Queries in High Dimensions," *Information Processing Letters,* vol. 45, no. 2, pp. 95-99, Feb. 1993.
[27] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: towards Removing the Curse of Dimensionality," *Proc. 30th Ann. ACM Symp. Theory of Computing,* pp. 604-613, May 1998.
[28] G.R. Hjaltason and H. Samet, "Incremental Similarity Search in Multimedia Databases," Computer Science Dept. TR-4199, Univ. of Maryland, College Park, Nov. 2000.
[29] T. Seidl and H.-P. Kriegel, "Optimal Multi-Step k-Nearest Neighbor Search," *Proc. ACM SIGMOD Conf.,* L. Hass and A. Tiwary, eds., pp. 154-165, June 1998.
[30] G.R. Hjaltason and H. Samet, "Ranking in Spatial Databases," *Proc. Advances in Spatial Databases-Fourth Int'l Symp.,* M.J. Egenhofer and J.R. Herring, eds., pp. 83-95, Aug. 1995.
[31] G.R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems,* vol. 24, no. 2, pp. 265-318, June 1999. Also Computer Science TR-3919, Univ. of Maryland, College Park.
[32] N. Linial, E. London, and Y. Rabinovich, "The Geometry of Graphs and Some of Its Algorithmic Applications," *Proc. 35th IEEE Ann. Symp. Foundations of Computer Science,* pp. 577-591, Nov. 1994.
[33] M. Linial, N. Linial, N. Tishby, and G. Yona, "Global Self Organization of All Known Protein Sequences Reveals Inherent Biological Signatures," *J. Molecular Biology,* vol. 268, no. 2, pp. 539-556, May 1997.
[34] L.J. Cowen and C.E. Priebe, "Randomized Non-Linear Projections Uncover High-Dimensional Structure," *Advances in Applied Math.,* vol. 19, pp. 319-331, 1997.
[35] A. Faragó, T. Linder, and G. Lugosi, "Fast Nearest-Neighbor Search in Dissimilarity Spaces," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 9, pp. 957-962, Sept. 1993.
[36] J. Vleugels and R.C. Veltkamp, "Efficient Image Retrieval through Vantage Objects," *Pattern Recognition,* vol. 35, no. 1, pp. 69-80, Jan. 2002.
[37] J.E. Barros, J. French, W. Martin, P.M. Kelly, and T.M. Cannon, "Using the Triangle Inequality to Reduce the Number of Comparisons Required for Similarity-Based Retrieval," *Proc. SPIE, Storage and Retrieval of Still Image and Video Databases IV,* I.K. Sethi and R. Jain, eds., vol. 2670, pp. 392-403, Jan. 1996.
[38] L. Mico, J. Oncina, and E. Vidal, "A New Version of the Nearest-Neighbour Approximating and Eliminating Search Algorithm (AESA) with Linear Preprocessing-Time and Memory Requirements," *Pattern Recognition Letters,* vol. 15, no. 1, pp. 9-17, Jan. 1994.
[39] M. Shapiro, "The Choice of Reference Points in Best-Match File Searching," *Comm. ACM,* vol. 20, no. 5, pp. 339-343, May 1977.
[40] E. Vidal Ruiz, "An Algorithm for Finding Nearest Neighbours in (Approximately) Constant Average Time," *Pattern Recognition Letters,* vol. 4, no. 3, pp. 145-157, July 1986.
[41] T.L. Wang and D. Shasha, "Query Processing for Distance Metrics," *Proc. 16th Int'l Conf. Very Large Databases,* D. McLeod, R. Sacks-Davis, and H.-J. Schek, eds., pp. 602-613, Aug. 1990.
[42] K.W. Pettis, T.A. Bailey, A.K. Jain, and R.C. Dubes, "An Intrinsic Dimensionality Estimator from Near-Neighbor Information," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 1, no. 1, pp. 25-37, 1979.
[43] X. Wang, J.T.L. Wang, K.-I. Lin, D. Shasha, B.A. Shapiro, and K. Zhang, "An Index Structure for Data Mining and Clustering," *Knowledge and Information Systems,* vol. 2, no. 2, pp. 161-184, May 2000.
[44] Y. Yang, K. Zhang, X. Wang, J.T.L. Wang, and D. Shasha, "An Approximate Oracle for Distance in Metric Spaces," *Proc. Ninth Ann. Symp. Combinatorial Pattern Matching,* M. Farach-Colton, ed., pp. 104-117, July 1998.
[45] K. Zhang, personal communication (unpublished), July 2000.

Gísli R. Hjaltason received the PhD degree in computer science from the University of Maryland, College Park, in 2000. From 2000 to 2002, he was a member of the technical staff of RightOrder, Inc., a Silicon Valley startup company, where he was involved in the research and development of an indexing technology with applications for relational and semistructured data. Since Fall 2002, he has been an assistant professor at the University of Waterloo, Canada. His research interests include database systems, indexing and query support for diverse data types (including spatial, multimedia, temporal, and semistructured data), geographical information systems (GIS), location-based services, and computational biology. He is a member of the IEEE Computer Society and the ACM.

Hanan Samet received the BS degree in engineering from the University of California, Los Angeles, and the MS degree in operations research and the MS and PhD degrees in computer science from Stanford University, Stanford, California. He is a fellow of the IEEE, ACM, and IAPR (International Association for Pattern Recognition). In 1975, he joined the Computer Science Department at the University of Maryland, College Park, where he is now a professor. He is a member of the Computer Vision Laboratory of the Center for Automation Research and also has an appointment at the University of Maryland Institute for Advanced Computer Studies. At the Computer Vision Laboratory, he leads a number of research projects on the use of hierarchical data structures for geographic information systems (GIS). His research group has developed the QUILT system that is a GIS based on hierarchical spatial data structures such as quadtrees and octrees, the SAND system which integrates spatial and nonspatial data, the SAND Browser which enables browsing through a spatial database using a graphical user interface, the VASCO set of JAVA applets (http://www.cs.umd.edu/hjs/quadtree/index.html) which demonstrate a wide range of spatial data structures and operations, and a symbolic image database system. Dr. Samet's research interests are data structures, computer graphics, geographic information systems (GIS), computer vision, robotics, pattern recognition, programming languages, artificial intelligence, and database management systems. He is the author of the books *The Design and Analysis of Spatial Data Structures*, and *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, both published by Addison-Wesley in 1990. He is an area editor of *Graphical Models*, and on the editorial board of *Image Understanding*, *Journal of Visual Languages*, *Pattern Recognition*, *GeoInformatica*, and *Journal of Spatial Cognition and Computation*.