CS-TR-4102 IRI-97-12715
January 2000

Contractive Embedding Methods for Similarity Searching in Metric Spaces

Gísli R. Hjaltason and Hanan Samet

Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742-3275
grh@cs.umd.edu and hjs@cs.umd.edu

Abstract

Complex data types (e.g., images, documents, DNA sequences, etc) are becoming increasingly important in database applications. The term *multimedia database* is often used to characterize such databases. A typical query for such data 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 of 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, such 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, our focus is on embedding methods that allow returning the same query result as if the actual distances of the objects are consulted, thus ensuring that no relevant objects are left out (i.e., there are no false dismissals). Particular attention was paid to SparseMap, a variant of Lipschitz embeddings, and FastMap, which is designed to be a heuristic alternative to the KLT (and the equivalent PCA and SVD) method for dimensionality reduction. We show that neither SparseMap nor FastMap guarantee that queries on the embedded objects have no false dismissals. However, we describe a variant of SparseMap allows queries with no false dismissals. Moreover, we show that with FastMap, 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 to guarantee no false dismissals.

This work was supported in part by the National Science Foundation under Grant IRI-97-12715.

1 Introduction

Multimedia databases are becoming increasingly important for storing and retrieving data in a wide range of applications including such fields as computational biology, computer aided design (CAD), image processing, etc. Examples of this type of data includes images, video, and text documents, and even such exotic data as protein and DNA sequences. A common type of queries on multimedia data is known as similarity searching (also termed *content-based* or *similarity* retrieval), and 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 with the target object (range query) or some given number of the most similar objects (nearest neighbor query).

1.1 Embedding

The level of similarity (or, actually, dis-similarity) between two objects is typically measured with a distance function d. Since the data objects in multimedia databases are usually of a complex nature, the distance function is often very expensive to compute. For example, computing the similarity of two proteins, i.e., based on their amino acid sequences, has been reported as taking several hundred milliseconds on typical workstations [19]. For this reason, it is desirable to make as few distance calculations as possible when executing similarity queries, and preferably none. A common approach to achieve this goal is to map the objects into a points in a low-dimensional space and then conduct the search there with the help of multidimensional indexing methods [21]. In general, the mapping must be made solely in terms of the distance function d rather than using information about the specific nature of the objects (i.e., d is a 'black box'). Most mapping methods require that d be a distance metric, i.e., that d satisfies the triangle inequality (or at least that only a relative small number of distances violate the triangle inequality).

In some multimedia applications, the mapping of data objects into vectors[1] is based on some specific *features* of the objects. Each coordinate axis in the vector space represents one of the features, and the data objects are termed feature vectors [8]. In image data, for example, a feature may represent the color in a portion of an image, or some aspect of the shape of the object being depicted [4, 8]. The distance function d is then usually defined with some weighted combination of the features that make up the feature vectors. This article is concerned with the general case when the data objects are arbitrary objects rather than feature vectors. Nevertheless, some of the techniques we discuss may also be relevant for the case when objects are represented by feature vectors since the dimensionality of the feature vectors may be too high to index effectively. Nevertheless, we do occasionally refer to the result of mapping general objects into a vector space as a feature vector, even though the coordinate axes do not really correspond to "features" of the object in this case.

1.2 General Distance Metrics

(S, d) is said to be a *finite metric space* if S is a finite set of size N and $d : S \times S \rightarrow \mathbf{R}^+$ is a distance metric. A great deal of work has been done on embedding finite metric spaces into low-dimensional real-normed spaces; i.e., real-valued vector spaces with a norm (i.e., a measure of the length of a vector) which serves as the basis of a distance metric. Such embeddings have been extensively

[1] We use the terms vectors and points interchangeabley as for our purposes these terms essentially have the same meaning.

studied in pure mathematics [3, 14] (where the embedding is often performed into Hilbert spaces, which are essentially abstractions of real-valued vector spaces with a dot product [25]), and have found application in a variety of settings [5, 12, 16, 19, 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*. 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 with d_E, d_A, and d_M, respectively. An important special case of metric space embedding is the one where the original finite metric space is a Euclidean space (i.e., a vector space with the Euclidean norm), or a vector space with another norm, of higher dimensionality than the embedding space (e.g., the original vectors might be feature vectors). This type of metric space embedding is referred to as *dimensionality reduction*.

Formally, an embedding of a finite metric space (S, d) into $(\mathbf{R}^k, d')$ is a mapping $F : S \rightarrow \mathbf{R}^k$, where k is the dimensionality of the embedding space and $d' : \mathbf{R}^k \times \mathbf{R}^k \rightarrow \mathbf{R}^+$ is the distance metric of the embedding space. If we denote the norm in $\mathbf{R}^k$ with $||\cdot||$, the distance metric d' is defined as $d'(x, y) = ||x \Leftrightarrow y||$. Ideally, the distance $d'(F(o_1), F(o_2))$ in the embedding space adheres closely to the distance $d(o_1, o_2)$ in the original space. However, it is often not possible and/or impractical to achieve exact correspondence between the distances based on d and d'. If an embedding F exists such that $d'(F(o_1), F(o_2)) = d(o_1, o_2)$ for all $o_1, o_2 \in S$, then (S, d) and $(\mathbf{R}^k, d')$ are said to be *isometric* (strictly speaking, (S, d) is isometric to $(F(S), d')$ where $F(S) \subset \mathbf{R}^k$ is the range of S).

For similarity retrieval applications, it is often useful to consider a larger, potentially infinite, metric space (U, d), where $S \subset U$, and define the domain of F as U rather than S. As a concrete example, U might be the set of all possible protein sequences, while $S \subset U$ is a particular data set under study. The reason for generalizing F is that, typically, the query object $q \in U$ used for similarity queries is not a member of S. Furthermore, in many applications, the database is dynamic, so that objects may be inserted into it and removed from it over time. Note that even though F can be applied on the entire set 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 be entirely suitable for objects in $U \setminus S$, 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. Rather than recompute F each time that the database S is updated, it is usually better to modify F only infrequently. The reason is that if F is recomputed from scratch, the value of $F(o)$ may change for every existing object $o \in S$. This recomputation would require rebuilding any index that has been built on the embedding space, which is usually a very costly process. One alternative possibility is to recompute the embedding F only when its quality (based on some measure, see Section 2.2) falls below some threshold.

1.3 Similarity Searching

Unless the distance measured with the distance function d' in the embedding space corresponds 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 shouldn't 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%, all the objects in R_E are correct, However, note that a precision value of 100% does not necessarily mean that R_E contains every element of the correct result. What it means is that none of the elements in R_E is not in the correct result. On the other hand, when the recall is 100%, all correct objects occur in R_E (as well as possibly some that are not!).

There are two ways to make use of R_E, the result from a query in the embedding space. One way 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 way 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%. However, unless the recall of queries in the embedding space is 100%, it is impossible to achieve 100% recall even with refinement (unless we resort to the expensive strategy of restarting the query in the embedding space with less selective criteria). In other words, if there are objects in R_O that are missing in R_E (i.e., R_E has less than 100% recall), then they will also be missing in R_F.

1.4 Outline

In this paper, our focus is on identifying the characteristics of embeddings F and/or the distance function d' in the embedding space that make it possible to achieve 100% recall. The rest of this paper is organized as follows. Section 2 discusses the properties of an embedding F, specifically ones that allow 100% recall, and methods for measuring the quality of F. Next, we survey a number of existing embedding methods. In particular, we describe multidimensional scaling (Section 3), Lipschitz embedding (Section 4) which also includes SparseMap, and FastMap (Section 5). Most of these embedding methods are applicable for general metric spaces, but some are dimension reduction methods, i.e., are specific to Euclidean space, or other normed-vector spaces. The explanation of some of the methods is quite detailed since this is necessary to see their limitations as this information was not obvious from their original presentation (e.g., for SparseMap and FastMap). Section 6 describes for each of the embedding methods whether (or under what conditions) they have the properties that result in 100% recall, and suggest modifications to the embedding if relevant. Section 7 contains concluding remarks and directions for future research.

2 Properties of Embeddings

In this section, we first describe basic properties of embeddings and why we want to perform an embedding (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). What we find is that the property that an embedding is *contractive* is sufficient to guarantee 100% recall of queries in the embedding space. How to make an embedding satisfy this property is the subject of the subsequent discussion in Section 6.

2.1 Basic Properties

As mentioned in Section 1, embedding multimedia 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 every pair 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 sufficiently high value of k, so that the distances between the points are approximately preserved when using a distance function d' in the k-dimensional space. In other words, for any pair of objects i and j, we have $d(i,j) \approx d'(F(i), F(j))$. 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 fast to calculate (i.e., require $O(N)$ or $O(N \log N)$ distance computations for the N objects instead of $O(N^2)$ distance computations), should preserve distances to a reasonable extent, and provide a fast way of obtaining the k-dimensional point corresponding to a query object (usually an object not in S).

At times, the mapping F can be chosen so that the distances between the objects are preserved exactly by F — that is, $d(i,j) = d'(F(i), F(j))$. For example, this is possible when the data objects are originally drawn from a vector space, and d and d' are both Euclidean distance metrics. In that particular case, distance preservation among the N objects is ensured when $k = N \Leftrightarrow 1$ or sometimes even for lower values of k. However, usually the distance values cannot be preserved exactly for an arbitrary combination of d and d', regardless of the value of k (i.e., there is no guarantee that a distance-preserving mapping F exists).

For example, suppose that we are given four objects a, b, c, and e with a distance function d such that the distance between each pair in $\{a, b, c\}$ is 2, while the distance from e to each of a, b, and c is 1.1. This distance function d satisfies the triangle inequality. However, these four objects cannot be embedded into a three-dimensional Euclidean space (i.e., d' is the Euclidean distance metric). In other words, we cannot position the objects in a three-dimensional space so that the Euclidean distance d' between the positions corresponds to the distance between the objects given by d. On the other hand, if the distance between e and the three remaining objects is at least $2/\sqrt{3}$, then such a positioning is possible by placing a, b, and c in a plane p and placing e on the line perpendicular to p that passes through the centroid of the triangle in p formed by a, b, and c.

Interestingly, the above embedding can be achieved if we use the City Block distance metric (L_1). In particular, this is the case when we position objects a, b, and c at locations (0,0,0), (2,0,0), and (1,1,0), respectively, and e at (1,0,0.1). This example illustrates that we may often obtain better distance correspondence by being flexible in choosing the distance function d'. In fact, it is always possible to achieve exact distance preservation when d' is the Chessboard metric (L_∞). In one such embedding, there is one dimension for each object o_i, where $o_1, o_2, \ldots, o_N$ is an enumeration of the objects. Each object o is mapped by F into the vector $\{d(o,o_1), d(o,o_2), \ldots, d(o,o_N)\}$. For any pair of objects o_i and o_j, their distance in the embedding is $d'(F(o_i), F(o_j)) = d_M(F(o_i), F(o_j)) = \max\{|F(o_i) \Leftrightarrow F(o_j)|\} = \max_l\{|d(o_i,o_l) \Leftrightarrow d(o_j,o_l)|\}$. Observe that for any l, $|d(o_i,o_l) \Leftrightarrow d(o_j,o_l)| \leq d(o_i,o_j)$ by the triangle inequality, while equality is reached for $l = i$ and $l = j$, and thus distances are indeed preserved by F when using the Chessboard metric. Notice that the number of dimensions in this distance preserving embedding is rather high, i.e., $k = N$. Thus, another choice of d' may be preferred if we want lower values of k, at the risk of lower quality in the embedding as distances may not be preserved.

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., [19]) is frequently used for this purpose. Distortion measures how much larger or smaller the distances in the embedding space $d'(F(o_1), F(o_2))$ are with respect to the corresponding distances $d(o_1, o_2)$ in the original space. In particular, the distortion is defined as $c_1 c_2$ when we are guaranteed that

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

for all pairs of objects $o_1, o_2 \in S$, where $c_1, c_2 \geq 1$. In other words, the distance values $d'(F(o_1), F(o_2))$ in the embedding spaces 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 a given embedding procedure, there may be no upper or lower bound on the distance ratio for the embeddings that it constructs, 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 results are known in general about embeddings, e.g., that any finite metric space can be embedded in Euclidean space with $O(\log N)$ distortion [19].

Another common measure of a particular embedding F with respect to a data set S is *stress* [16]. Stress measures the overall deviation in the distances, and is typically defined as

$$\frac{\sum_{o_1, o_2} (d'(F(o_1), F(o_2)) \Leftrightarrow d(o_1, o_2))^2}{\sum_{o_1, o_2} d(o_1, o_2)^2}.$$

Alternative definitions of stress may be more appropriate for certain applications. For example, the sum in the denominator may be on $d'(F(o_1), F(o_2))^2$, 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 [12] is termed Cluster Preservation Ratio (CPR). This measure can be applied on data sets 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 the s neighbors. For an example where the clustering of the objects is known, consider the situation of proteins in a computational biology application [12]. In particular, a number of proteins have been studied extensively in terms of their biochemical function, so proteins having a similar function can be grouped together. Therefore, we can test whether amino acid sequences representing these known proteins follow this grouping.

Finally, we also want to point out that precision and recall (as defined in Section 1) 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%, but poor distance preservation will lower 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 on 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 map F is said to be *contractive* if $d'(F(o_1), F(o_2)) \leq d(o_1, o_2)$ for all $o_1, o_2 \in S$. In other words, $c_2 = 1$ in Equation 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. For example, consider a range query with a radius of r with respect to object q, where we wish to identify objects $o \in S$ such that $d(q, o) \leq r$. Given that the embedding is contractive, we are ensured that $d(q, o) > r$ if $d'(F(q), F(o)) > r$. Thus, we can safely prune all objects o from the search for which $d'(F(q), F(o)) > r$ without any *false dismissals* — that is, no relevant object is dropped from the query result, and thus we have 100% recall. For this reason, we sometimes refer to the contractive property as the *pruning property*.

Another useful, but rarely satisfied, property of a mapping F is *proximity preservation*, i.e., the property that $d(o_1, o_2) \leq d(o_1, o_3) \Rightarrow d'(F(o_1), F(o_2)) \leq d'(F(o_1), F(o_3))$. 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 d'. Thus, we 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 $c_1 c_2$ (see Equation 1):

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

In other words, $d'(F(o_1), F(o_2))$ can be no more than a factor of $c_1 c_2$ larger than $d'(F(o_1), F(o_3))$. 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 $d'(F(q), F(o'))$ can be smaller than $d'(F(q), F(o))$ by as much as a factor of $c_1 c_2$, 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 $c_1 c_2$. In many applications, exact results to nearest neighbor queries are not crucial and approximate results are satisfactory, at least if the error is not too high (e.g., see [1, 2, 13]). 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 (variants that allow a small error in the result are also possible if we wish to trade off accuracy for possible increase in efficiency). Such algorithms use a filter and refine strategy [9, 15, 22] (as described in Section 1). 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 $d'(F(q), F(o')) \leq d(q, o)$. 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 distance from $F(q)$ in the embedding space. When using a multidimensional index, this can be achieved by using an incremental nearest neighbor algorithm [10, 11]. Suppose that point $F(a)$ corresponding to object a is the closest point to

$F(q)$ at a distance of $d'(F(a), F(q))$. We compute the distance $d(a, q)$ between the corresponding objects. At this point, we know that any point farther from $F(q)$ than $d(a, q)$ cannot correspond to the nearest neighbor of q, since the contractive property guarantees that $d'(F(x), F(q)) > d(a, q)$ means that $d(x, q) > d(a, q)$ for any object x. 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 distance $d'(F(x), F(q))$ greater than the distance to the current closest object, which is now guaranteed to be the actual closest object to q.

3 Multidimensional Scaling

Multidimensional scaling [16, 24] (MDS) is a method of constructing an embedding F that works for arbitrary metric spaces. The method has been widely used for many decades, in both the social and physical sciences, for the purpose of visualizing and clustering the data resulting from experiments and studies, as well as other purposes. MDS is defined in many ways, some of which even allow non-metric distances (i.e., satisfaction of the triangle inequality is not required). One of the more common variants seeks to minimize *stress*, as defined in Section 2.2, i.e.,

$$\frac{\sum_{o_1,o_2}(d'(F(o_1), F(o_2)) \Leftrightarrow d(o_1, o_2))^2}{\sum_{o_1,o_2} d(o_1, o_2)^2}.$$

Minimizing stress is essentially a non-linear optimization problem, where the variables are the $N \cdot k$ coordinate values corresponding to the embedding (i.e., k coordinate values for each of the N objects). Typically, solving such a problem involves starting with an arbitrary assignment of the variables, and then trying to improve on it in an iterative manner using the method of steepest descent (e.g., [16]). The result of the optimization is not always the embedding that obtains the absolute minimum stress, but instead it is one that achieves a local minimum (i.e., the minimization can be pictured as finding the deepest valley in a landscape by always walking in a direction that leads downhill; the process can thus get stuck in a deep valley that is not necessarily the deepest).

In principle, it is possible to make multidimensional scaling result in a contractive embedding, by constraining the minimization of the stress with $O(N^2)$ contractive conditions, one for each pair of objects. Unfortunately, multidimensional scaling has a limited applicability in similarity search, regardless of whether the resulting embedding is contractive or not. This is partly due to the high cost of constructing the embedding, both in terms of the number of distance computations (i.e., $O(N^2)$, one for each pair of objects) and due to the inherent complexity of the optimization process. More seriously, when performing similarity queries, we must compute the embedding of the query object q, again, by minimizing stress subject only to varying the coordinate values of $F(q)$. Although the minimization process is itself expensive, the most serious drawback is that the distances of all objects in S from q must be computed in order to evaluate the stress. This completely defeats the goal of performing similarity queries in terms of the embedding space, namely that of avoiding as many distance computations as possible. In fact, after computing the distances of all objects in S from q, we can immediately tell what the result of the query should be, thereby making the embedding of q unnecessary.

4 Lipschitz Embeddings

A powerful class of embedding methods is known as Lipschitz embeddings [3, 14]. They are based on defining a coordinate space where each axis corresponds to a reference set which is a subset of the objects. This is the subject of this section, which is organized as follows. Section 4.1 contains a definition of a Lipschitz embedding. Section 4.2 describes how to select the reference sets. Section 4.3 explains SparseMap, which is an instance of a Lipschitz embedding that attempts to reduce the computational cost of the embedding. Section 4.4 presents an example Lipschitz embedding.

4.1 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. Notice that the distance preserving L_∞ embedding that we described in Section 2.1 is a special case of a Lipschitz embedding, where R consists of all singleton subsets of S (i.e., $R = \{\{o_1\}, \{o_2\}, \ldots, \{o_N\}\}$).

The intuition behind the 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 by comparing $d(o_1, x)$ and $d(o_2, x)$, i.e., the value $|d(o_1, x) \Leftrightarrow d(o_2, x)|$. This is especially true if one of the distances $d(o_1, x)$ and $d(o_2, x)$ is small. Observe that due to the triangle inequality we have $|d(o_1, x) \Leftrightarrow d(o_2, x)| \leq d(o_1, o_2)$, as illustrated in Figure 1. This argument can be extended to a subset A. In other words, the value $|d(o_1, A) \Leftrightarrow d(o_2, A)|$ 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 $|d(o_1, A) \Leftrightarrow d(o_2, A)| = |d(o_1, x_1) \Leftrightarrow d(o_2, x_2)|$. Accounting for the case that $d(o_1, x_1) \Leftrightarrow d(o_2, x_2)$ is positive or negative, we have that $|d(o_1, x_1) \Leftrightarrow d(o_2, x_2)| \leq \max\{|d(o_1, x_1) \Leftrightarrow d(o_2, x_1)|, |d(o_1, x_2) \Leftrightarrow d(o_2, x_2)|\}$. Finally, from the triangle inequality we have that $\max\{|d(o_1, x_2) \Leftrightarrow d(o_2, x_2)|, |d(o_1, x_1) \Leftrightarrow d(o_2, x_1)|\} \leq d(o_1, o_2)$. Thus $|d(o_1, A) \Leftrightarrow d(o_2, A)|$ 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., $d'(F(o_1), F(o_2))$).

4.2 Selecting Reference Sets

With a suitable definition of R, the set of reference sets, we can establish bounds on the distance $d'(F(o_1), F(o_2))$ for all pairs of objects $o_1, o_2 \in S$, where d' is one of the L_p metrics. Such a definition was provided by Linial, London, and Rabinovich [18, 19], based in part on previous work by Bourgain [3]. In particular, in their definition [19], 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 $\lfloor \log_2 N \rfloor$ (or perhaps $\lfloor \log_2(N \Leftrightarrow 1) \rfloor$). 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 $j = \lfloor (i \Leftrightarrow 1)/(\log_2 N) + 1 \rfloor$. The embedding proposed by Linial et al [19] is a variant of the basic Lipschitz embedding, where each coordinate value is divided by a factor that depends on k. In particular, if d' is the L_p metric,

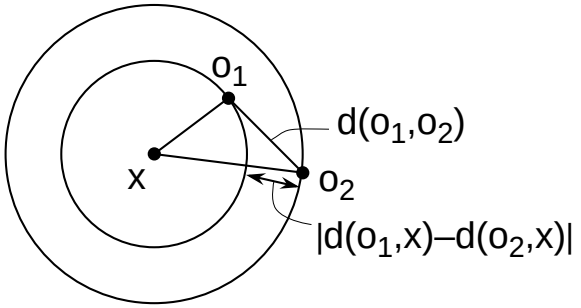

Figure 1: Demonstration of the distance bound $|d(o_1, x) \Leftrightarrow d(o_2, x)| \leq d(o_1, o_2)$. The objects o_1, o_2, and x are represented as points, and the distance between them with the length of the line segments between them.

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, London and Rabinovich [19] prove that F satisfies

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

for any pair of objects $o_1, o_2 \in S$, where $c > 0$ is a constant[2]. Thus, the distortion in distance values, i.e., the relative amount of deviation of the distance values in the embedding space with respect to the original distance values, is guaranteed to be $O(\log N)$ (with high probability). The proof for the bound $c/\lfloor \log_2 N \rfloor d(o_1, o_2) \leq d'(F(o_1), F(o_2))$ is rather sophisticated [3, 19], and is beyond the scope of this paper. However, the bound $d'(F(o_1), F(o_2)) \leq d(o_1, o_2)$ is easy to show. In particular, for each $A_i \in R$, we have $|d(o_1, A_i) \Leftrightarrow d(o_2, A_i)| \leq d(o_1, o_2)$, as shown in Section 4.1. Thus, when d' is an arbitrary L_p distance metric,

$$d'(F(o_1), F(o_2)) = \left(\sum_{i=1}^{k} \left(\frac{d(o_1, A_i) \Leftrightarrow d(o_2, A_i)}{k^{1/p}} \right)^p \right)^{1/p} \tag{3}$$

$$\leq \left(k \cdot \frac{d(o_1, o_2)^p}{k} \right)^{1/p} = d(o_1, o_2) \tag{4}$$

A distortion of $O(\log N)$ may seem to be rather large, and may render the embedding ineffective at preserving relative 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 [12, 17], the embedding defined above was found to lead to good preservation of clusters, as defined by biological functions of proteins.

Notice that the mapping F as defined by Linial et al [19] is contractive (i.e., satisfies the pruning property), which is advantageous for the purpose of similarity search. In many other situations, only relative differences in distances are important in the embedding space, while the contractive property is immaterial. In other words, the crucial property that we wish to retain is which objects are close to each other and which are far (i.e., a weak form of proximity preservation). An example of applications of this sort is cluster analysis [12, 17]. In such situations it may be more convenient to use the regular Lipschitz embedding definition of F with respect to the set of reference sets R

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

defined in [19] (i.e., without dividing by $k^{1/p}$). Recall that $k = \lfloor \log_2 N \rfloor^2$ thereby implying that $\sqrt{k} = \lfloor \log_2 N \rfloor$. Therefore, when the Euclidean distance metric is being used, this embedding guarantees distance bounds of

$$c \cdot d(o_1, o_2) \leq d_E(F(o_1), F(o_2)) \leq \lfloor \log_2 N \rfloor \cdot d(o_1, o_2)$$

for any pair of objects $o_1, o_2 \in S$, where $c > 0$ is a constant (with high probability).

Unfortunately, the embedding of [19] 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 in S, which is exactly what we wish to avoid. Second, the number of subsets in R, and thus the number of coordinate values (i.e., dimensions) in the embedding, is relatively large — that is, $\lfloor \log_2 N \rfloor^2$. Even with as few as 100 objects, the number of dimensions is 36, much too high to index on efficiently with multidimensional indexing methods. These drawbacks were acknowledged in [19], but addressing them was left for future work (the only suggestion that was made was to drop the sets A_i of largest sizes).

4.3 SparseMap

SparseMap [12] is an embedding method originally proposed for mapping a database of proteins into Euclidean space. It is built on the work of Linial et al [19] in that the same set of reference sets R is used. The SparseMap method [12] comprises two heuristics, each aimed at alleviating one of the drawbacks discussed in Section 4.2 — 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 4.2. 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 [12] 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 [19] (i.e., 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 guarantees with respect to 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 6 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 the $d(o, A_1)$ for all objects $o \in S$, next $d(o, A_2)$ for all objects o, etc. Since evaluating $d(o, A_i)$ for any given object o can be very expensive in terms of distance computations, SparseMap adopts a heuristic that instead computes $\hat{d}(o, A_i)$, which is an upper bound on $d(o, A_i)$. This heuristic 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 i^{th} 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 [12]. Greedy resampling is applied after the entire k coordinate axes have already been determined, and its goal is to reduce the number of coordinate axes down to $k' < k$. Essentially, this means eliminating of 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 [16] 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 5). 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., 4000 in experiments in [12], which constituted 10% 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 have been obtained.

In order to study the validity of the SparseMap method, various experiments are presented in [12], 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 [5], another embedding method proposed for similarity searching described in greater detail in Section 5. 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 this is the number of iterations used when trying to determine the two objects having the greatest distance from each other. Thus, the two methods can be made to perform approximately the same number of distance computations for obtaining a given number of coordinate axes. In the experiments reported in [12], when this is done, the embedding produced by SparseMap is of a higher quality than that of FastMap, in terms of the stress as well as how well clusters are retained (these clusters were defined by the biological function of the proteins). Furthermore, for a small number of coordinate axes, the embedding produced by SparseMap was of higher quality (as defined above) than that produced by FastMap. In addition, SparseMap was found to scale up better than FastMap, in terms of the time to perform the mapping, as the pattern in which the database is accessed leads to fewer disk I/Os.

4.4 Example

Figure 2 shows the inter-object distances between ten objects (these distance values were constructed by positioning ten two-dimensional points and measuring the L_1 distance between them). In this case, $\lfloor \log_2 N \rfloor = 3$, so we could have three reference sets of each of three different sizes (2, 4, and 8), for a total of 9 dimensions. Since a set of size 8 contains nearly all the objects, we instead choose to use only reference sets of sizes 2 and 4, and two sets of each size. Choosing objects at random, we arrive at the sets $A_1 = \{o_2, o_8\}$, $A_2 = \{o_1, o_5\}$, $A_3 = \{o_6, o_8, o_9, o_{10}\}$, and $A_4 = \{o_1, o_4, o_7, o_8\}$. The resulting four-dimensional coordinates are given in Figure 3. Here, we

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
o_1	0	2	13	7	3	8	11	4	9	10
o_2	2	0	11	9	3	10	9	2	11	8
o_3	13	11	0	6	10	9	4	9	6	3
o_4	7	9	6	0	6	3	8	9	2	5
o_5	3	3	10	6	0	7	8	3	8	7
o_6	8	10	9	3	7	0	9	10	3	6
o_7	11	9	4	8	8	9	0	7	10	3
o_8	4	2	9	9	3	10	7	0	11	6
o_9	9	11	6	2	8	3	10	11	0	7
o_{10}	10	8	3	5	7	6	3	6	7	0

Figure 2: Distance matrix for ten sample objects.

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
A_1	2	0	9	9	3	10	7	0	11	6
A_2	0	2	10	6	0	7	8	3	8	7
A_3	4	2	3	2	3	0	3	0	0	0
A_4	0	2	4	0	3	3	0	0	2	3

Figure 3: Four-dimensional coordinate values for the ten objects based on the distances in Figure 2, as determined by a Lipschitz embedding with the four reference sets $A_1 = \{o_2, o_8\}$, $A_2 = \{o_1, o_5\}$, $A_3 = \{o_6, o_8, o_9, o_{10}\}$, and $A_4 = \{o_1, o_4, o_7, o_8\}$.

use the regular Lipschitz embedding (which is used in SparseMap) where coordinate value j for object i is $d(o_i, A_j)$, rather than $d(o_i, A_j)/k^{1/p}$ as specified by Linial, London and Rabinovich [19]. For example, $d(o_4, A_2) = \min\{d(o_4, o_1), d(o_4, o_5)\} = \min\{7, 6\} = 6$.

We now give an example of how to compute the distance between two objects in the embedding space, when d' is the Euclidean distance metric. In particular, for objects o_3 and o_8 we have from Figure 3 that $F(o_3) = (9, 10, 3, 4)$ and $F(o_8) = (0, 0, 3, 0)$. Therefore, $d'(F(o_3), F(o_8)) = \sqrt{(9 \Leftrightarrow 0)^2 + (10 \Leftrightarrow 3)^2 + (3 \Leftrightarrow 0)^2 + (4 \Leftrightarrow 0)^2} = \sqrt{81 + 49 + 9 + 16} = \sqrt{155} \approx 12.4$. In comparison, their actual distance is $d(o_3, o_8) = 9$. Notice that in this case the distance in the embedding space is greater than their actual distance. In contrast, when using the embedding of Linial et al [19], the distance in the embedding space would have been about $12.4/\sqrt{4} = 6.2$. Also, if d' had been the Chessboard distance metric (L_∞), the distance in the embedding space would have been $\max\{9, 7, 3, 4\} = 9$, which happens to equal $d(o_3, o_8)$.

5 FastMap

FastMap [5] is designed to provide a heuristic alternative to dimensionality reduction methods for Euclidean space that are based on linear transformations. The Karhunen-Loève transform (KLT) [6], as well as the equivalent principal component analysis (PCA) [6] and the singular value decomposition (SVD) [7] methods, are widely used examples of such methods. In this section we review the motivation (Section 5.1) for its development, outline how it works (Sections 5.2–5.6), and give an example of its usage (Section 5.7). In particular, Section 5.2 explains the general

principles behind FastMap by outlining how the coordinate axes that make up the mapping are constructed. Section 5.3 describes how the pivot objects that anchor the lines that form the newly-formed coordinate axes are chosen. Section 5.4 shows how the first coordinate value is determined. Section 5.5 presents the modified distance function for computing the remaining coordinate values, while Section 5.6 applies this modified distance function to actually compute the remaining coordinate values.

5.1 Motivation

For a set S of points in a m-dimensional Euclidean space, the Karhunen-Loève transform (KLT) [6], as well as the equivalent principal component analysis (PCA) [6] and the singular value decomposition (SVD) [7] methods, identify a new set of m coordinate axes, represented by unit length vectors $V = \{v_1, v_2, \ldots, v_m\}$, termed basis vectors. The origin of the new coordinate system is taken to be the center of gravity for the points in S, and the new coordinate values are obtained by projecting each point in S onto the basis vectors. Observe that this amounts to a linear transformation of the set S, involving translation and rotation. The basis vectors are ordered in decreasing order of the variance along them, so that v_1 has the most variance, then v_2, etc. The variance along an axis indicates how spread out the projections of the data points on the axis are, and is equal to the sum of the squared coordinate values along the axis (this definition of variance assumes that the mean of the coordinate values is zero, which is the case here). Loosely speaking, the set V is chosen such that the variance is as great as possible along each basis vector in turn. More precisely, for any other basis $W = \{w_1, w_2, \ldots, w_m\}$, there exists some i such that the variance is greater along v_i than w_i, while the variance is equal for v_j and w_j where $j < i$ (of course, for $j > i$, the variance along w_j may be greater).

Once the set S has been linearly transformed in the manner described above, it is easy to reduce the dimensionality of the points in S to k by dropping all but the first k coordinate values of the transformed points. Since the variance along the first k coordinate axes is as great as possible, this results in the least loss of distance information, in terms of the mean square error (i.e., the sum of the squares of the Euclidean distances between each m-dimensional point and its corresponding k-dimensional point). To be more precise, KLT (as well as the equivalent PCA and SVD) yields the linear transformation with minimum mean square error. Some non-linear transformations, e.g., one produced by multidimensional scaling (see Section 3), may lead to a smaller error. For example, if all the points lie on curved line in two-dimensional space, then a non-linear transformation may exist that leads to perfect distance preservation when reducing the dimensionality to one, while this would not be the case for any linear transformation. On the other hand, if the points lay on a straight line, a linear transformation would indeed exist that leads to perfect distance preservation.

Notice that these transformation methods are only applicable for vector spaces, and not arbitrary metric spaces, since they are inherently defined in terms of matrices composed of the vector data (i.e., the matrix has the size $N \times m$, where N is the number of vectors and m is the number of dimensions). Moreover, they are only really meaningful if d is the Euclidean distance metric, and not for the other Minkowski metrics. The reason is twofold. First, the other Minkowski metrics are not invariant under rotation. In other words, the distance between some pairs of points may increase or decrease, depending on the direction of the rotation. Second, variance is defined in terms of second powers, just like the Euclidean distance metric. Thus, the variance criteria that determine the rotation, and in turn what axes to drop when reducing the dimensionality, are inherently related to the Euclidean distance metric. In particular, the sum of the variance over all coordinate axes corresponds to the sum of the squared Euclidean distances from the origin to each

point (recall that we are assuming that the mean along each dimension is at the origin). Therefore, dropping the axes having the least variance corresponds to reducing as little as possible the sum of the squared Euclidean distances from the origin to each point.

FastMap is an attempt to generalize the principles of the KLT (as well as the equivalent PCA and SVD) method for obtaining embeddings of arbitrary metric spaces (rather than just of Euclidean spaces) into k-dimensional Euclidean space. Besides the motivation of generalizing KLT (and the equivalent PCA and SVD) 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 a more accurate assessment of the execution time complexity of the FastMap method is $O(Nmk)$ in this setting. Nevertheless, as we shall see, in the remainder of this section as well as in Section 6, this generalization is not without its limitations.

5.2 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 Euclidean space of some dimensionality, or in other words, that d is the Euclidean distance metric[3]. Nevertheless, FastMap can be applied with varying success with other distance metrics. In particular, as we will see in Section 6, 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 small as just 1)[4]. Similarly, due to the nature of the FastMap method, the best result is obtained when d', the distance function in the embedding space, is the Euclidean distance metric. Thus, unless otherwise stated, we assume d' 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 [5] 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 formed through 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, without loss of generality, 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 obtain the next coordinate axis by determining the $(m \Leftrightarrow 1)$-dimensional hyperplane H that is perpendicular to the line that forms the previous coordinate axis, and project 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 d_H is derived

[3] More precisely, the assumptions made by FastMap are valid if (S, d) is isometric to 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.

[4] Of course, this would not be a problem if the "intrinsic" dimensionality [20] of the data is low, but this need not be the case.

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 have been determined.

Figure 4 helps to illustrate how the first coordinate axis is determined, and how the distance function d_H is used to determine the second coordinate axis. Figure 4a 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_i for object i (i.e., the first coordinate value in the vector $F(i)$) is the distance from r to the projection of i onto the line through r and s. We postpone for now the discussion of Figure 4b, which illustrates how d_H and the next set of coordinate values is determined.

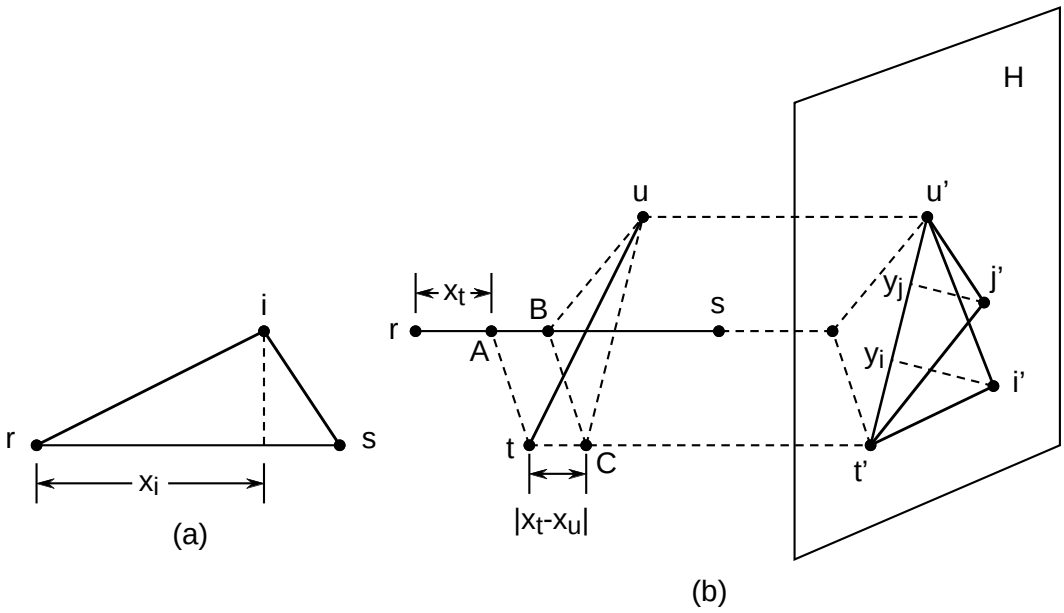

Figure 4: Examples of projections of objects with the FastMap method on (a) the first coordinate axis and (b) the second coordinate axis.

5.3 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. Ideally, there should be a large spread of the projected values on the line between the pivot objects, where *spread* is defined as $\max_{i,j} |x_i \Leftrightarrow x_j|$. The reason is that a greater spread generally means that more distance information can be extracted from the projected values, i.e., more information can be gleaned from $|x_i \Leftrightarrow x_j|$ for any pair of objects i and j. This principle is similar to that used in the KLT (as well as the equivalent SVD and PCA) method, described in Section 5.1. The difference, here, is that spread along an axis is a weaker notion than that of variance, which is used by KLT (as well as the equivalent SVD and PCA). The reason why spread is a weaker notion is that a large spread may be caused by a few outliers while most of the other values may be clustered within a small range. On the other hand, a large variance genuinely means that the values are widely scattered around a large range. Nevertheless, spread usually provides a reasonably good estimate of the level of variance. In order to maximize the likelihood of obtaining a large spread, the pivot objects should be as far as possible from each other. Unfortunately, determining the furthest pair of objects among a given set of N objects is

computationally expensive. In particular, it takes $O(N^2)$ distance computations as we need to examine the distance between each pair of objects.

Faloutsos and Lin [5] propose a heuristic for computing an approximation of the furthest pair of objects. This heuristic first arbitrarily chooses one of the objects i. Next, it finds the object r which is furthest from i. Finally, it finds the object s which is the furthest from r. The last step can be iterated a number of times (e.g., 5 [5]) in order to obtain a better estimate of the pair that is the furthest apart. In fact, it can be shown that for a given set of N objects, the procedure for finding the furthest pair of objects can be iterated a maximum of $N \Leftrightarrow 1$ 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 6.2. Note that the original distance function d is used only when determining the first coordinate axis. However, the modified distance functions (i.e., 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 for any objects i and j we have $d(i,j) \leq 2d(r,s)$, or $d(r,s) \geq \frac{1}{2}d(i,j)$. In other words, $d(r,s)$ is at least half of the distance between the most distant pair of objects. This follows directly from the triangle inequality and the assumption that s is the object farthest from r. In particular, $d(i,j) \leq d(r,i) + d(r,j) \leq d(r,s) + d(r,s) = 2d(r,s)$. However, this bound is only guaranteed to hold for the first pair of pivot objects, as shown in Section 6.2, since the distance functions used to determine the second and subsequent coordinate values may not satisfy the triangle inequality. Notice that a tighter lower bound for $d(r,s)$ cannot be guaranteed regardless of the number of iterations in the heuristic, unless the number of iterations is $N \Leftrightarrow 1$. As an example, one pair of objects may have a distance of 2 while all other pairs have a distance of 1. Determining the pair having a distance of 2 requires $O(N^2)$ distance computations on the average, since the number of distinct pairs is $O(N^2)$.

5.4 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 Figure 5 for an object i. Note that the projection of an object i may actually lie beyond the line segment between r and s as shown in Figure 5b. This does not pose problems, but may cause x_i to be negative. The actual value of x_i is obtained by solving the following equation for x_i:

$$d(r,i)^2 \Leftrightarrow x_i^2 = d(s,i)^2 \Leftrightarrow (d(r,s) \Leftrightarrow x_i)^2. \tag{5}$$

Expanding terms in 5 and rearranging yields:

$$x_i = \frac{d(r,i)^2 + d(r,s)^2 \Leftrightarrow d(s,i)^2}{2d(r,s)}. \tag{6}$$

Observe that Equation 5 is obtained by applying the Pythagorean theorem on each half of the triangle in Figure 5a (a similar interpretation applies to the case in Figure 5b). Since the Pythagorean theorem is specific to Euclidean space, we have here an instance where Faloutsos and Lin [5] 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

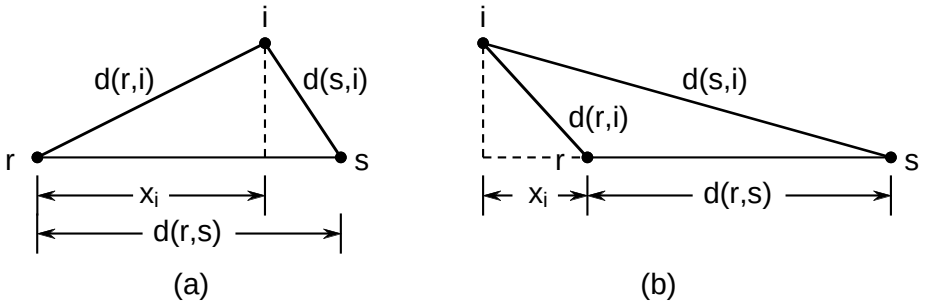

Figure 5: Examples of two possible positions for the projection of an object on the line joining the points corresponding to the pivot objects.

implications of the heuristic are explored in Section 6.2; 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_i, based on Equation 6 and the selection of pivot objects. First, $x_r = 0$ and $x_s = d(r,s)$, as would be expected. Second, note that $|x_i| \leq d(r,s)$, implying that the maximum difference between two values x_i and x_j (i.e., the spread for the first coordinate axis as defined earlier) is $2d(r,s)$, which is equal to the maximum possible distance between any pair of objects as shown in Section 5.3. In fact, it can be shown that the spread is never larger than the distance between the farthest pair of objects. Since the spread is at least $d(r,s)$ when r and s serve as the pivot objects (as $x_r = 0$ and $x_s = d(r,s)$), this implies that the spread obtained by pivots r and s is at least half of the maximum obtainable spread which is $2d(r,s)$. Below, we show that no choice of other pair of pivot objects can have a larger spread.

Lemma 1 *Let t and u be the two objects that are farthest apart for all the objects in S. The spread is never more than $d(t,u)$ for any choice of pivot objects.*

Proof Assume that $d(t,u) = \alpha d(r,s)$ where we observe that $1 \leq \alpha \leq 2$ when r and s are chosen according to the heuristic presented in Section 5.3. An upper bound on the value of x_i for any object i is easily derived given the upper bound of $d(r,s)$ for $d(r,i)$ and the lower bound of 0 on $d(s,i)$:

$$\begin{aligned} x_i &= \frac{d(r,i)^2 + d(r,s)^2 \Leftrightarrow d(s,i)^2}{2d(r,s)} \\ &\leq \frac{2d(r,s)^2}{2d(r,s)} = d(r,s). \end{aligned}$$

To determine a lower bound on x_i, we appeal to the triangle inequality, which yields $|d(r,s) \Leftrightarrow d(s,i)| \leq d(r,i)$. Raising both sides to the second power yields

$$(d(r,s) \Leftrightarrow d(s,i))^2 = d(r,s)^2 + d(s,i)^2 \Leftrightarrow 2d(r,s)d(s,i) \leq d(r,i)^2.$$

In other words, we have

$$d(r,i)^2 \Leftrightarrow d(s,i)^2 \geq d(r,s)^2 \Leftrightarrow 2d(r,s)d(s,i) \geq (1 \Leftrightarrow 2\alpha)d(r,s)^2$$

since $d(s,i) \leq d(t,u) = \alpha d(r,s)$. Substituting this lower bound into Equation 6 yields

$$x_i = \frac{d(r,i)^2 + d(r,s)^2 \Leftrightarrow d(s,i)^2}{2d(r,s)}$$

$$\geq \quad \frac{(2 \Leftrightarrow 2\alpha)d(r,s)^2}{2d(r,s)} = (1 \Leftrightarrow \alpha)d(r,s).$$

Combining the upper and lower bounds on the value of x_i and the fact that $\alpha \leq 2$ yields the desired bound $|x_i| \leq d(r,s)$. Furthermore, for any $\alpha \geq 1$ we can derive the maximum spread as

$$|x_i \Leftrightarrow x_j| \leq d(r,s) \Leftrightarrow (1 \Leftrightarrow \alpha)d(r,s) = \alpha d(r,s) = d(t,u).$$

Thus, we are guaranteed that the maximum spread is at most $d(t,u)$ for any choice of pivots, as desired. ■

Unfortunately, as we show in Section 6.2, the distance functions used in subsequent iterations of FastMap may not satisfy the triangle inequality if d is not the Euclidean distance metric. Thus, the above bounds may not hold when determining the subsequent coordinate values.

5.5 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 on the hyperplane H, as mentioned in Section 5.2. Figure 4b illustrates how d_H is formed and 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 $|x_t \Leftrightarrow x_u|$. 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 deg. Therefore, we have

$$d_H(t',u')^2 = d(u,C)^2 = d(t,u)^2 \Leftrightarrow d(t,C)^2 = d(t,u)^2 \Leftrightarrow (x_t \Leftrightarrow x_u)^2. \tag{7}$$

Note that this equation applies to any pair of objects t and u and not just the ones that serve as pivots in the next iteration, as is the case in Figure 4b.

Observe that this is another occasion where Faloutsos and Lin [5] in their development of the method make the implicit assumption that d is the Euclidean distance metric (or behaves like one, i.e., if (S,d) is isometric to a Euclidean space). This assumption has some undesirable side-effects. For example, as we show in Section 6.2, if d is not a Euclidean distance metric, then d_H may fail to satisfy the triangle inequality, which in turn may cause Equation 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 $d_H(i',j')^2$. 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 i and j (or, more precisely, their projections) cannot serve as pivot objects.

5.6 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 Equations 6 and 7 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 i^{th} iteration of FastMap. In particular, x_o^i is the i^{th} coordinate value 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 i^{th} 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 Equation 6 for iteration i is

$$x_o^i = \frac{d_i(p_1^i, o)^2 + d_i(p_1^i, p_2^i)^2 \Leftrightarrow d_i(p_2^i, o)^2}{2d_i(p_1^i, p_2^i)}, \tag{8}$$

given the recursive distance function definition

$$\begin{aligned} d_1(a,b) &= d(a,b) \\ d_i(a,b)^2 &= d_{i-1}(a,b)^2 \Leftrightarrow (x_a^{i-1} \Leftrightarrow x_b^{i-1})^2 \\ &= d(a,b)^2 \Leftrightarrow d_E(F_{i-1}(a), F_{i-1}(b))^2. \end{aligned} \tag{9}$$

Notice that in this presentation of the projected distance functions, we use the original objects as arguments, rather than using a' and b', as is done in Equation 7 (i.e., that equation uses t' and u' for the projections of t and u, respectively).

The process of mapping the N objects to points in a k-dimensional space takes $O(k \cdot N)$ 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 needs a $2 \times k$ array to record the identity 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 it could be done if we don't 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 (which is actually $O(1)$ if we assume that k is a constant) in contrast to the size of the database which is $O(N)$.

5.7 Example

Referring back to the table in Figure 2, we now show how FastMap obtains two-dimensional coordinate values for the objects. Seeing that the largest distance in the table is between o_1 and o_3, we choose these objects as pivots. The result is shown as the first dimension in Figure 6 (i.e., in the first row), where the values are given to a precision of one fractional digit. As an example of how these values are determined, we derive the first coordinate value of o_5:

$$\frac{d(o_1, o_5)^2 + d(o_1, o_3)^2 \Leftrightarrow d(o_3, o_5)^2}{2d(o_1, o_3)} = \frac{3^2 + 13^2 \Leftrightarrow 10^2}{2 \cdot 13} = 78/26 = 3.$$

Figure 7 shows the (squared) projected distances $d_H(o_i, o_j)^2$ for the ten objects obtained by projecting them on the hyperplane H perpendicular to the line through the pivot objects o_1 and

Dim.	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
1	0.0	2.0	13.0	7.0	3.0	5.8	10.5	4.0	8.2	10.0
2	4.4	1.0	4.4	8.5	3.7	9.7	0.7	0.0	10.2	2.8

Figure 6: Two-dimensional coordinate values for the ten objects based on the distances in Figure 2, as determined by FastMap using o_1 and o_3 as pivot objects for the first coordinate axis, and o_8 and o_9 for the second coordinate axis.

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
o_1	0.0	0.0	0.0	0.0	0.0	29.8	9.9	0.0	13.3	0.0
o_2	0.0	0.0	0.0	56.0	8.0	85.2	8.1	0.0	82.2	0.0
o_3	0.0	0.0	0.0	0.0	0.0	29.8	9.9	0.0	13.3	0.0
o_4	0.0	56.0	0.0	0.0	20.0	7.7	51.5	72.0	2.5	16.0
o_5	0.0	8.0	0.0	20.0	0.0	40.9	7.2	8.0	36.6	0.0
o_6	29.8	85.2	29.8	7.7	40.9	0.0	59.0	96.6	3.3	18.7
o_7	9.9	8.1	9.9	51.5	7.2	59.0	0.0	6.2	94.7	8.7
o_8	0.0	0.0	0.0	72.0	8.0	96.6	6.2	0.0	103.1	0.0
o_9	13.3	82.2	13.3	2.5	36.6	3.3	94.7	103.1	0.0	45.9
o_{10}	0.0	0.0	0.0	16.0	0.0	18.7	8.7	0.0	45.9	0.0

Figure 7: Distances of the ten sample objects as determined by the first projected distance function, d_H. The values in the table are actually the squared distances, $d_H(o_i, o_j)^2$.

o_3. Again, the distance values are only given to a precision of one fractional digit. As an example of how these distance values are computed, we derive $d_H(o_5, o_6)^2$ with the help of Figure 6:

$$d_H(o_5, o_6)^2 = d(o_5, o_6)^2 \Leftrightarrow |x_5 \Leftrightarrow x_6|^2 \approx 7^2 \Leftrightarrow |3 \Leftrightarrow 5.8|^2 = 49 \Leftrightarrow 2.8^2 \approx 41.2.$$

This value does not exactly match the value $d_H(o_5, o_6)^2 \approx 40.9$ found in Figure 7 due to round off error.

The largest distance value is $d_H(o_8, o_9)^2 \approx 103.1$, so the objects o_8 and o_9 (or, more precisely, their projected versions) get chosen as the second pair of pivot objects used to determine the values along second dimension as given in Figure 6. Again, let us show how the coordinate value for o_5 is determined, this time along the second coordinate axis:

$$\frac{d_H(o_8, o_5)^2 + d_H(o_8, o_9)^2 \Leftrightarrow d_H(o_9, o_5)^2}{2 d_H(o_8, o_9)} \approx \frac{8 + 103.1 \Leftrightarrow 36.6}{2\sqrt{103.1}} \approx 74.5/20.3 \approx 3.7.$$

5.8 Heuristic for Non-Euclidean Metrics

When d is not a Euclidean metric, the value $d_H(t', u')^2$ in Equation 7 may be negative, as mentioned above. More generally, for the formulation in Equation 9, 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, since $d_i(a, b)^2$ can become a large negative value, such values can cause a large distortion in the distances between coordinate values determined by Equation 8, as detailed in Section 6.2.3.

In [23], a heuristic was introduced for alleviating 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:

$$d_i(a,b) = \begin{cases} \sqrt{d_{i-1}(a,b)^2 \Leftrightarrow (x_a^{i-1} \Leftrightarrow x_b^{i-1})^2}, \text{ if } d_{i-1}(a,b)^2 \geq (x_a^{i-1} \Leftrightarrow x_b^{i-1})^2, \\ \Leftrightarrow\sqrt{(x_a^{i-1} \Leftrightarrow x_b^{i-1})^2 \Leftrightarrow d_{i-1}(a,b)^2}, \text{ otherwise.} \end{cases} \quad (10)$$

Equivalently, we can use the definition $d_i(a,b) = \text{sign}(d_i(a,b)^2) \cdot \sqrt{|d_i(a,b)^2|}$, where $d_i(a,b)^2$ is defined as in Equation 9. Although this heuristic apparently resolves the drawbacks of negative $d_i(a,b)^2$ values, it does not correct the fundamental problem with Equation 9, namely the fact that d_i may violate the triangle inequality if d_{i-1} is not the Euclidean distance metric. Furthermore, notice that this formulation also means that $d_i(a,b)^2 = d(a,b)^2 \Leftrightarrow d_E(F_{i-1}(a), F_{i-1}(b))^2$ no longer holds if $d_j(a,b)$ is negative for $2 \leq j \leq i$.

Notice that when $d_i(a,b)$ is defined according to Equation 10, the value of $d_i(a,b)^2$ is always non-negative, regardless of whether $d_i(a,b)$ is negative or not. Thus, in situations where Equation 9 leads to negative values, the coordinate value x_o^i for an object o as determined by Equation 8 can be different depending on which definitions of d_i is used (i.e., Equations 9 or 10). 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 (i.e., it is sometimes better to use Equation 9 and sometimes better to use Equation 10). However, the advantage of the definition in Equation 10 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, Equation 10 implies that $d_i(a,b)^2 = |d_{i-1}(a,b)^2 \Leftrightarrow (x_a^{i-1} \Leftrightarrow x_b^{i-1})^2|$, so $d_i(a,b)^2$ is only larger than $d_{i-1}(a,b)^2$ if $(x_a^{i-1} \Leftrightarrow x_b^{i-1})^2 > 2d_{i-1}(a,b)^2$. In contrast, according to Equation 9, the value of $d_i(a,b)^2$ is monotonically non-increasing in i, so it can become a large negative value. In Section 6.2.3, 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 Equation 9, such pairs could not be utilized. However, it is not clear how appropriate such a choice is, in terms of resulting in good distance preservation. Furthermore, 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 pivots further increases the distance distortion in $d'(F(a), F(b))$.

6 Contractive Property

In this section, we show that the embeddings resulting from SparseMap (Section 6.1) and FastMap (Section 6.2) are not contractive. More importantly, we demonstrate that a contractive embedding can be obtained by modifying one of the heuristics of SparseMap. Unfortunately, FastMap cannot be modified to always result in a contractive embedding.

6.1 SparseMap

A drawback of the embedding that forms the basis of SparseMap (i.e., the regular Lipschitz embedding on the reference sets, without taking the heuristics into account) is that it is not contractive, and thus fails to satisfy the pruning property. 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 [19] can be employed (i.e., where the coordinate values are divided by $k^{1/p}$), which is indeed contractive. Second, the

distance function d' 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 $d'(F(o_1), F(o_1)) = d_p(F(o_1), F(o_2))/(k^{1/p})$. The advantage of modifying the distance function d' rather than the embedding itself is that it allows modifying the number of coordinate axes (which occurs, 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 Equation 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 (in particular, bounds that rely on N and/or k) 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 $|\hat{d}(o_1, A_i) \Leftrightarrow \hat{d}(o_2, A_i)|$ 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 be larger than the actual distances $d(o_1, A_i)$ and $d(o_2, A_i)$ (which are the minimum distances from o_1 and o_2 to an object in A_i) by an arbitrary amount. In particular, we cannot preclude a situation where $\hat{d}(o_1, A_i) > \hat{d}(o_2, A_i) + d(o_1, o_2)$, in which case $|\hat{d}(o_1, A_i) \Leftrightarrow \hat{d}(o_2, A_i)| > d(o_1, o_2)$.

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 thereof 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 d' in the embedding space. Note that $d_M(F(o_1), F(o_2)) \leq d(o_1, o_2)$ as shown in Section 4.2 (the key observation is that $|d(o_1, A_i) \Leftrightarrow d(o_2, A_i)| \leq d(o_1, o_2)$ 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 $d(o, x)$. Observe that in SparseMap, the approximate distances $d_E(F_{i-1}(o), F_{i-1}(x))$ are computed for each $x \in A_i$, which has the same cost complexity as evaluating $d_M(F_{i-1}(o), F_{i-1}(x))$, although the constants of proportionality are lower for d_M than for d_E. 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 computed so far following this procedure. Once all lower bound distances $d_M(F_{i-1}(o), F_{i-1}(x))$ of the remaining elements $x \in A_i$ are greater than $d(o, y)$, we are assured that $d(o, A_i) = d(o, y)$.

Even though we described our modified heuristic in terms of the Chessboard distance metric, by using a suitable definition of the distance function d' 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 d' based on L_p is defined as

$$d'(F_{k'}(o_1), F_{k'}(o_2)) = \frac{(\sum_i |d(o_1, A_i) \Leftrightarrow d(o_2, A_i)|^p)^{1/p}}{(k')^{1/p}}. \tag{11}$$

For any choice of p, this distance metric makes F contractive (e.g., note the similarity with Equation 4).

Moreover, observe that for fixed values of o_1, o_2, and $F_{k'}$, the function d' as defined by Equation 11 increases with increasing values of p. For example, for $p = 1$, $d'(F_{k'}(o_1), F_{k'}(o_2))$ 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 $d'(F_{k'}(o_1), F_{k'}(o_2))$ 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 $d'(F(q), F(o')) \leq r$, then o' will be a part of the result set when performing a query in the embedding space. Thus, the situation can easily arise that $d'(F(q), F(o')) \leq r$ when basing d' on the City Block or Euclidean distance metrics (i.e., L_1 or L_2) but $d'(F(q), F(o')) > r$ when d' is based on the Chessboard distance metric L_∞. Such a hypothetical example is illustrated in Figure 8.

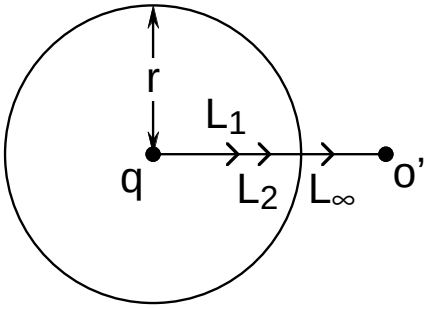

Figure 8: A hypothetical range query example where an object o' is outside the distance range r from the query object q. The distance $d'(F(q), F(o'))$ from q to o' in the embedding space will lie somewhere on the line between q and o'. Thus, this distance may lie inside the query range if d' is based on L_1 or L_2, but outside if d' is based on L_∞.

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 pre-processing) may be justified as the resulting embedding is contractive. This allows effective pruning in similarity queries, while obtaining accurate results as we get 100% recall and thus do not miss any relevant answers.

6.2 FastMap

If the metric space (S, d) is actually a Euclidean space (or isometric to a Euclidean space), then the embedding F produced by FastMap is contractive (Section 6.2.1). Unfortunately, if (S, d) is not isometric to any Euclidean space, then the embedding produced by FastMap is no longer guaranteed to be contractive (Section 6.2.2). In fact, it is more likely than not that the embedding is not contractive, at least for a few pairs of objects. 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 6.2.3). 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. In addition, failure to satisfy the contractive property also reduces the extent to which distances can be preserved (Section 6.2.4).

6.2.1 Contractiveness of FastMap

The embedding F produced by FastMap is contractive when the metric space (S, d) is actually a Euclidean space (or isometric to a Euclidean space). This is not surprising since key aspects of FastMap are based on a property unique to Euclidean spaces, namely the Pythagorean theorem. In particular, both Equation 6, that computes a single coordinate value, and Equation 7, that computes the projected distance d_H used in the next iteration, are based on the Pythagorean theorem. To see why contractiveness is satisfied, we point out that FastMap can be used to extract m coordinate values for each object from the distance function d, assuming that the vectors in S are m-dimensional[5].

Each coordinate value extracted by FastMap is equivalent to rotating and translating the set of data points in such a way that the pivot objects have a certain alignment along the given coordinate axis. Furthermore, note that the projected distance function d_H used to determine the second pair of pivots measures distances along a hyperplane H that is perpendicular to the line through the first pair of pivot objects r and s. Therefore, the line through r and s is orthogonal to the line through the second pair of pivot objects (which are defined in terms of d_H), and by extension, all subsequent pairs of pivot objects. The same argument can be applied recursively to the other pairs of pivot objects, thereby showing that the coordinate axes are all mutually orthogonal. Notice that $d' = d$, since both are Euclidean distance metrics for m-dimensional space. Thus, we see that the result of FastMap is equivalent to applying a linear transformation involving translation and rotation, just like in KLT as discussed in Section 5, which are distance-preserving operations under the Euclidean distance function (i.e., $d(o_1, o_2) = d(F(o_1), F(o_2))$). Hence, if we omit some of the m coordinate values, the distances among the resulting vectors can only be smaller, and thus the mapping is contractive.

6.2.2 Non-Contractiveness of FastMap

Figure 9 illustrates how FastMap would map objects i and j having determined just one coordinate value, assuming that r and s are the pivot objects. Notice how it is intuitively obvious that the distance from $F(i)$ to $F(j)$ is smaller than that between i and j. Unfortunately, intuition is misleading, here, as it tends to be based on Euclidean geometry, and we perceive the three-dimensional world around us as obeying Euclidean geometry. In particular, the relative lengths of the line segments between points in Figure 9 can only arise if d is the Euclidean distance metric (or if (S, d) is isometric to a Euclidean space). Thus, we see that in the figure, $d(i, j)^2 = (x_i \Leftrightarrow x_j)^2 + D^2$ (according to the Pythagorean theorem), so we clearly have $d(i, j) \geq |x_i \Leftrightarrow x_j|$. In general, we cannot assume that this relationship holds.

The satisfaction of the contractive property is only guaranteed when both d and d' are the Euclidean distance metric (i.e., the original objects are points in a multidimensional space). There are two general scenarios where the contractive property does not hold. The first takes place when d is the Euclidean distance metric while d' is a non-Euclidean distance metric (Lemma 2). The second takes place when d is not the Euclidean distance metric (and (S, d) is not isometric to a Euclidean space), regardless of the nature of d' (Lemmas 3–5), and is due to the implicit assumption in Equations 6 and 7 that d is the Euclidean distance metric. The first scenario is captured in the following lemma.

[5] Actually, if the vectors in S lie on a m'-dimensional subspace, $m' < m$, then FastMap will only be able to extract m' coordinate values. However, in this case, the missing coordinate values can simply be set to 0 (if they are indeed desired).

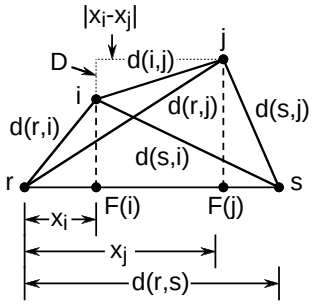

Figure 9: Example projection of two objects on the line joining the points corresponding to the pivot objects.

Lemma 2 *If d is a Euclidean metric and d' is some Minkowski metric L_p, the embedding F produced by FastMap is only guaranteed to be contractive if $p \geq 2$.*

Proof We showed in Section 6.2.1 that FastMap leads to a contractive embedding if d' is the Euclidean metric (i.e., L_2). Let $x = \{x_1, \ldots, x_m\}$ and $y = \{y_1, \ldots, y_m\}$ be some arbitrary points in S. The L_p distance between x and y, $d_p(x, y) = (\sum_{i=1}^{m} |x_i \Leftrightarrow y_i|^p)^{1/p}$, can be shown to be non-increasing as a function of p. Thus, if d_A is the L_1 metric, then $d_A(F(o_1), F(o_2)) \geq d_E(F(o_1), F(o_2))$, so we may have $d_A(F(o_1), F(o_2)) > d(o_1, o_2) \geq d_E(F(o_1), F(o_2))$. Therefore, is not guaranteed to be contractive if d' is L_1 (i.e., $p = 1$). This situation cannot arise if d' is some other Minkowski metrics L_p where $p \geq 2$ since $d'(F(o_1), F(o_2)) \leq d_E(F(o_1), F(o_2)) \leq d(o_1, o_2)$ for such d'. ■

The second, and more serious, scenario of contractive property violation occurs when d is not a Euclidean metric, regardless of the choice of d'. The reason why this is serious is that the contractive property may be violated after determining any number of coordinate axes. In particular, this means that the value of $d_i(a, b)^2$ in Equation 9 can be negative (or zero) for any $i \geq 2$, which precludes using a and b as the pivot pair in iteration i of FastMap. Thus, the situation can arise that $d_i(a, b)^2 \leq 0$ for all pairs of objects a and b, in which case no more coordinate values can be determined. This dilemma is avoided by using the heuristic discussed in Section 5.8, but possibly at the price of even more distortion of the distance values.

There are two possible causes for the violation of the contractive property: Equation 6 and Equation 7. Both are due to the implicit assumption made in deriving these equations that the Pythagorean theorem applies to the distance metric d. However, the Pythagorean theorem is unique to Euclidean spaces.

Lemma 3 *Let (S, d) be a finite metric space that is not isometric to a Euclidean space. In one iteration of FastMap, Equation 6 may result in values that violate the contractive property.*

Proof As an example where Equation 6 causes the contractive property to be violated, consider objects a, b, c, and e, with the following distance matrix for which satisfaction of the triangle inequality can be easily verified:

	a	b	c	e
a	0	10	4	5
b	10	0	8	7
c	4	8	0	1
e	5	7	1	0

Since objects a and b are the objects which are the furthest apart, they are selected as the pivot objects in FastMap when determining the first coordinate axis. Following Equation 6, the values of the first coordinate of the points derived from objects c and e are as follows:

$$\begin{aligned} x_c &= (4^2 + 10^2 \Leftrightarrow 8^2)/(2 \cdot 10) = 52/20 = 13/5 \\ x_e &= (5^2 + 10^2 \Leftrightarrow 7^2)/(2 \cdot 10) = 76/20 = 19/5 \end{aligned}$$

Thus, the distance between these one-dimensional points corresponding to objects c and e used in the first iteration of FastMap is $x_e \Leftrightarrow x_c = 6/5 = 1.2$, which is higher than the original distance between them, obtained from the distance matrix, which is 1. Thus the contractive property does not hold for c and e. ■

The contractive property is not directly violated by Equation 7, but instead indirectly. In particular, if d is not a Euclidean distance metric, Equation 7 may lead to a definition of d_H such that d_H does not satisfy the triangle inequality (thereby failing to be a distance metric), and this, in turn, causes the pruning property to be violated in the next iteration of FastMap.

Lemma 4 *Let (S, d) be a finite metric space that is not isometric to a Euclidean space. The distance function d_H as defined by Equation 7 (which uses the values computed in Equation 6) may not satisfy the triangle inequality. In other words, it is possible for d_H to not be a distance metric.*

Proof Consider objects a, b, c, and e, with the following distance matrix for which satisfaction of the triangle inequality can be easily verified:

	a	b	c	e
a	0	6	5	4
b	6	0	3	4
c	5	3	0	6
e	4	4	6	0

Following Equation 6, the values of the first coordinate of the points derived from c and e using a and b as the pivot objects are obtained as follows:

$$\begin{aligned} x_c &= (5^2 + 6^2 \Leftrightarrow 3^2)/(2 \cdot 6) = 13/3 \\ x_e &= (4^2 + 6^2 \Leftrightarrow 4^2)/(2 \cdot 6) = 3 \end{aligned}$$

The projected distances between a', c', and e' follow from Equation 7:

$$\begin{aligned} d_H(a', c') &= \sqrt{5^2 \Leftrightarrow (0 \Leftrightarrow 13/3)^2} \approx 2.494 \\ d_H(a', e') &= \sqrt{4^2 \Leftrightarrow (0 \Leftrightarrow 3)^2} \approx 2.647 \\ d_H(c', e') &= \sqrt{6^2 \Leftrightarrow (13/3 \Leftrightarrow 3)^2} \approx 5.850 \end{aligned}$$

Thus we see that $d_H(a', c') + d_H(a', e') \approx 5.141$ which is less than $d_H(c', e')$, thereby violating the triangle inequality. ■

We next show how the violation of the triangle inequality causes the pruning property to be violated.

Lemma 5 *If d does not satisfy the triangle inequality, the contractive property may possibly not be satisfied.*

Proof Assuming that a and b are the pivot objects, the contractive property is violated if $d(a,i) < |x_a \Leftrightarrow x_i| = x_i$ (since $x_a = 0$) for some object i. Now, let us explore what conditions are equivalent to $d(a,i) < x_i$, and thus give rise to a violation of the contractive property.

$$\begin{aligned}
d(a,i) < x_i \quad &\Leftrightarrow \quad d(a,i) < \frac{d(a,i)^2 + d(a,b)^2 \Leftrightarrow d(b,i)^2}{2d(a,b)} \\
&\Leftrightarrow \quad 2d(a,i)d(a,b) < d(a,i)^2 + d(a,b)^2 \Leftrightarrow d(b,i)^2 \\
&\Leftrightarrow \quad d(b,i)^2 < d(a,i)^2 \Leftrightarrow 2d(a,i)d(a,b) + d(a,b)^2 = (d(a,i) \Leftrightarrow d(a,b))^2 \\
&\Leftrightarrow \quad d(b,i) < |(d(a,i) \Leftrightarrow d(a,b))| \\
&\Leftrightarrow \quad d(a,i) + d(b,i) < d(a,b) \vee d(a,b) + d(b,i) < d(a,i)
\end{aligned}$$

Similarly, it can be shown that $d(b,i) < |x_b \Leftrightarrow x_i|$ iff $d(a,i)+d(b,i) < d(a,b) \vee d(a,b)+d(a,i) < d(b,i)$. Thus, we see that the contractive property is violated (for $d(a,i)$, $d(b,i)$, or both) if and only if the triangle inequality is violated for any of the distances among the two pivot objects a and b and an arbitrary object i. ■

Thus, in Lemmas 3 through 5, we have established the following:

Theorem 1 *Let (S,d) be a finite metric space that is not isometric to a Euclidean space. Both Equation 6 and Equation 7 (and by extension, the general forms in Equations 8 and 9) may cause the contractive property to be violated.* ■

Observe that the non-Euclidean heuristic discussed in Section 5.8, embedded in Equation 10, does not affect this result, as the proofs did not make use of the fact that $d_H(i,j)^2$ is negative.

6.2.3 Large Distortion with FastMap

In the previous section, we showed that the embedding F produced by FastMap may fail to satisfy the contractive property. In this section, we are concerned with the question of 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 $\max_{o_1,o_2}\{\frac{d'(F(o_1),F(o_2))}{d(o_1,o_2)}\}$. Naturally, if the expansion is no more than 1, then F is contractive. 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 $d'(o_1,o_2) = d_E(F(o_1),F(o_2))/c$ such that the expansion with respect to this d' is no more than 1. Unfortunately, as we shall see, the expansion of embeddings produced by FastMap when determining just one coordinate is already relatively large, and for additional coordinates the expansion can be very large, especially if the heuristic discussed in Section 5.8 is not employed. Below, we first treat the case when the non-Euclidean heuristic is not used (Section 6.2.3.1), and then describe how our results are affected when using the heuristic (Section 6.2.3.2). This is followed by a brief summary (Section 6.2.3.3).

6.2.3.1 Original Approach

First, we derive an upper bound on the expansion when determining just one coordinate with FastMap.

Lemma 6 *The expansion after determining one coordinate with FastMap is at most 3. This bound is tight.*

Proof Using Equation 6, we can derive an upper bound on the expansion, with the help of the triangle inequality and the upper bound $d(r,s)$ on $d(r,i)$ and $d(r,j)$:

$$\begin{aligned}
\frac{|x_i \ominus x_j|}{d(i,j)} &= \left|\frac{d(r,i)^2 \ominus d(s,i)^2 \ominus d(r,j)^2 + d(s,j)^2}{2d(r,s)d(i,j)}\right| \\
&= \left|\frac{(d(r,i) \ominus d(r,j))(d(r,i)+d(r,j)) + (d(s,j) \ominus d(s,i))(d(s,j)+d(s,i))}{2d(r,s)d(i,j)}\right| \\
&\leq \frac{|d(r,i) \ominus d(r,j)|(d(r,i)+d(r,j)) + |d(s,j) \ominus d(s,i)|(d(s,j)+d(s,i))}{2d(r,s)d(i,j)} \\
&\leq \frac{d(i,j)(d(r,i)+d(r,j)) + d(i,j)(d(s,j)+d(s,i))}{2d(r,s)d(i,j)} \\
&= \frac{d(r,i)+d(r,j)+d(s,j)+d(s,i)}{2d(r,s)} \\
&\leq \frac{d(r,i)+d(r,j)+d(r,s)+d(r,j)+d(r,i)+d(r,s)}{2d(r,s)} \leq \frac{6d(r,s)}{2d(r,s)} = 3.
\end{aligned}$$

We now show that this bound is tight. Consider the following assignment of the distances among the objects i, j, r, and s, where $d(r,s)$ is left as a variable:

$$\begin{array}{lllllll}
d(r,i) & = & d(r,s) & & d(s,i) & = & (2 \ominus 2\alpha)d(r,s) \\
d(r,j) & = & (1 \ominus \alpha)d(r,s) & & d(s,j) & = & (2 \ominus \alpha)d(r,s) \\
d(i,j) & = & \alpha d(r,s) & & & &
\end{array}$$

The resulting distances clearly satisfy the triangle inequality for $0 < \alpha < 1$. Using these distance values leads to the following value of $|x_i \ominus x_j|$:

$$\begin{aligned}
|x_i \ominus x_j| &= \left|\frac{d(r,s)^2 \ominus (2 \ominus 2\alpha)^2 d(r,s)^2 \ominus (1 \ominus \alpha)^2 d(r,s)^2 + (2 \ominus \alpha)d(r,s)^2}{2d(r,s)}\right| \\
&= \left|\frac{1}{2}(1 \ominus 4 \ominus 4\alpha^2 + 8\alpha \ominus 1 \ominus \alpha^2 + 2\alpha + 4 + \alpha^2 \ominus 4\alpha)d(r,s)\right| \\
&= \left|\frac{1}{2}(\ominus 4\alpha^2 + 6\alpha)d(r,s)\right| = d(r,s)(3\alpha \ominus 2\alpha^2).
\end{aligned}$$

Thus, dividing by $d(i,j) = \alpha d(r,s)$ gives the expansion on $d(i,j)$, namely $3 \ominus 2\alpha$. Clearly, as α approaches 0, the expansion approaches the value of 3. ■

The example provided in the proof of the tightness of the bound uses an arbitrarily small value of $d(i,j)$. A more realistic assumption is that there is some bound on the ratio between any two distance values, e.g., that the largest distance value is no more than 10 times larger than the smallest distance value, or equivalently, the smallest distance value is no smaller than $d(r,s)/5$ (since the

largest distance value is at most $2d(r,s)$). However, even for such a small range of values, the distortion can be as much as $3 \Leftrightarrow 2 \cdot \frac{1}{5} = 2.6$ (actually, the ratio between the smallest and largest distance value in the example is then 9, as the smallest distance value is $d(i,j) = \frac{1}{5}d(r,s)$ while the largest distance value is $d(s,j) = (2 \Leftrightarrow \frac{1}{5})d(r,s) = \frac{9}{5}d(r,s)$).

As we showed in Section 6.2.2, the triangle inequality does not necessarily hold for the distance functions used in the second and subsequent iterations of FastMap. This means that the upper bound on the expansion derived in Lemma 6 only holds for the first coordinate value obtained by FastMap. The basic problem is that for iteration $i > 1$ the value of $d_i(r,s)^2$, as defined in Equation 9, may be less than or equal to zero for all objects s, even after several iterations of the pivot finding heuristic (see Section 5.3). Moreover, even when $d_i(r,s)^2$ is found to be strictly positive, it can be arbitrarily close to zero, thereby yielding an arbitrarily large expansion. One solution is to set a strict lower bound on the distance between the pivot objects, based on the distance between the first pair of pivot objects. In other words, 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, and the number of distance computations may become $O(N^2)$. 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 5.3). 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 preferrable for most applications. Nevertheless, it is still not clear that relative distances are adequately preserved 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 $d_2(t,u) \geq d(r,s)/\beta$ ($\beta > 1$), we now derive an upper bound on the expansion in the second iteration of FastMap, i.e., when finding the second coordinate value for each object.

Lemma 7 *The expansion after determining two coordinates with FastMap is no more than $36\beta^2$, where $d(r,s)/\beta$ is the minimum distance value for the pivots used in the second iteration (using the projected distance function) as well as for the distances between any two pairs of objects (using the original distance function), where $\beta > 1$ is some constant.*

Proof Below, we derive an upper bound for $\frac{|x_a^2 - x_b^2|}{d(a,b)}$, using the definitions from Section 5.6. Notice that the formula for $|x_a^2 \Leftrightarrow x_b^2|$ is symmetric in a and b, so we can use it without the absolute value operator. In the derivation, we make use of the lower bound $d(r,s)/\beta$ and the upper bound $2d(r,s)$ on any distance values. Furthermore, we use the bound obtained in Lemma 6. In particular, for any objects u and v we obtain

$$d_2(u,v)^2 = d(u,v)^2 \Leftrightarrow |x_u \Leftrightarrow x_v|^2 \geq d(u,v)^2 \Leftrightarrow (3d(u,v))^2 = \Leftrightarrow 8d(u,v)^2. \tag{12}$$

Thus, applying Equation 8 (with $i = 2$), the lower bound on $d_2(u,v)^2$ in Equation 12, its upper bound $d_2(u,v)^2 \leq d(u,v)^2$, and the bounds on the original distances, we obtain

$$\begin{aligned}
\frac{x_a^2 \Leftrightarrow x_b^2}{d(a,b)} &= \frac{d_2(p_1^2,a)^2 \Leftrightarrow d_2(p_2^2,a)^2 \Leftrightarrow d_2(p_1^2,b)^2 + d_2(p_2^2,b)^2}{2d_2(p_1^2,p_2^2)d(a,b)} \\
&\leq \frac{d(p_1^2,a)^2 \Leftrightarrow (\Leftrightarrow 8d(p_2^2,a)^2) \Leftrightarrow (\Leftrightarrow 8d(p_1^2,b)^2) + d(p_2^2,b)^2}{2(d(r,s)/\beta)(d(r,s)/\beta)} \\
&\leq \frac{4d(r,s)^2 + 8 \cdot 4d(r,s)^2 + 8 \cdot 4d(r,s)^2 + 4d(r,s)^2}{2d(r,s)^2/\beta^2}
\end{aligned}$$

$$= \frac{18 \cdot 4d(r,s)^2}{2d(r,s)^2/\beta^2} = \frac{72}{2/\beta^2} = 36\beta^2. \blacksquare$$

The upper bound in Lemma 7 is not tight. The reason is that the triangle inequality on the original distances and the pivot criteria for p_1^2 and p_2^2 prevent all the upper bounds utilized in the proof from being realized simultaneously. It is difficult to derive a tighter bound that takes into account all of these criteria. Nevertheless, we attempted to construct an example exhibiting large expansion, through the use of a non-linear optimization tool. The objective function describing the expansion involved 14 variables and the number of constraints (incorporating all the relevant criteria) was about 80. Due to this relatively large number, the optimizer frequently got stuck at local maxima rather than finding the true maximum. However, it was clear that the maximum expansion is at least proportional to β. For example, for $\beta = 5$ (which meant that the largest distance value was never more than ten 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.

A general formula for $k > 2$ coordinate values can be derived in a recursive fashion in a similar way as we derived Lemma 7, which would yield an upper bound of $O(\beta^{2(k-1)})$ for the expansion. Again, this upper bound is not attainable. By implementing a similar non-linear optimization model as we described above for $k = 3$, we obtained anecdotal evidence that the upper bound is $O(\beta^{k-1})$. This time, the model had 27 variables and over 200 conditions, and it yielded an expansion as large as 75 for $\beta = 2\frac{1}{2}$, and 340 for $\beta = 5$, a difference by a factor close to 4 when β was increased by a factor of 2. 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.

6.2.3.2 Non-Euclidean Heuristic

The upper bound of 3 on the expansion in the first coordinate value established in Lemma 6 also applies when the heuristic described in Section 5.8 is employed 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 5.8. 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 Equation 8. Without the heuristic (i.e., when using Equation 9), the value of $d_i(a,b)^2$ is monotonically non-decreasing in i. Thus, $d_i(a,b)^2$ can become a large negative value (i.e., much smaller than $\Leftarrow d(a,b)^2$), even if the expansion due to each individual iteration of FastMap is minor. In contrast, when using Equation 10, the value of $d_i(a,b)^2$ is always non-negative, and unless the expansion is more than 2 for coordinate axis i (i.e., $(x_a^{i-1} \Leftarrow x_b^{i-1})^2 > 2d_{i-1}(a,b)^2$), $d_i(a,b)^2$ is no larger than $d_{i-1}(a,b)^2$.

Since the expansion after the first iteration is at most 3, $d_2(a,b) \geq \Leftarrow\sqrt{9d(a,b)^2 \Leftarrow d(a,b)^2} = \Leftarrow\sqrt{8}d(a,b)$, so $0 \leq d_2(a,b)^2 \leq 8d(a,b)^2$ when using Equation 10. Proceeding as in Lemma 7, we can show that the expansion due to the second iteration of FastMap is no more than $32\beta^2$ (where $d(r,s)/\beta$, $\beta > 1$, is the minimum distance for the second pivot pair). This is nearly as much as when not using the heuristic (where we established the upper bound $36\beta^2$). However, as before, this upper bound is not attainable. Modifying the non-linear optimization model to account for the heuristic, we obtained an expansion of up to 7 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, unlike in the case without the heuristic (i.e., when $d_i(r,s)^2$ is negative). However, it is not obvious what effect it has on the quality of the mapping when $d_i(r,s)$ is negative for the pivot objects r and s, so it is not clear whether it is advisable to allow such pivot objects. Furthermore, if $|d_i(r,s)|$ is very small, then the expansion can potentially get very large. Thus, in order to keep the expansion from becoming too large, we still must be careful about how the pair of pivot objects is chosen.

6.2.3.3 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, thereby 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 the expansion are not often encountered. 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.

6.2.4 Distance Preservation

An interesting question is the extent to which FastMap preserves distances, i.e., when $d(a,b)$ equals $d'(F(a),F(b))$. Is it possible to achieve distance preservation for sufficiently large values of k? Below, we examine these questions, and find that distance preservation is possible when d is the Euclidean metric, but not otherwise. Recall that we assume that d' is the Euclidean distance metric. Unless otherwise mentioned, also assume below that Equation 9 is used to compute distances.

Lemma 8 shows that pivot objects are indistinguishable in subsequent iterations of FastMap. Thus, the pivot objects in iteration $i \Leftrightarrow 1$ have identical coordinate values for coordinate axes i through k.

Lemma 8 *Let r and s be the pivot objects in iteration $i \Leftrightarrow 1$ of FastMap. Then, $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for an arbitrary object t, regardless of whether or not d_{i-1} satisfies the triangle inequality.*

Proof The first part, $d_i(r,s) = 0$, is obvious, since

$$x_r^{i-1} = \frac{d_{i-1}(r,r)^2 + d_{i-1}(r,s)^2 \Leftrightarrow d_{i-1}(r,s)}{2d_{i-1}(r,s)} = 0$$

and

$$x_s^{i-1} = \frac{d_{i-1}(r,s)^2 + d_{i-1}(r,s)^2 \Leftrightarrow d_{i-1}(s,s)}{2d_{i-1}(r,s)} = d_{i-1}(r,s).$$

Now, let t be an arbitrary object. Using Equations 8 and 9, we can compute the values of $d_i(r,t)^2$ and $d_i(s,t)^2$:

$$\begin{aligned}
d_i(r,t)^2 &= d_{i-1}(r,t)^2 \Leftrightarrow |x_r^{i-1} \Leftrightarrow x_t^{i-1}|^2 = d_{i-1}(r,t)^2 \Leftrightarrow (x_t^{i-1})^2 \\
&= d_{i-1}(r,t)^2 \Leftrightarrow \left(\frac{d_{i-1}(r,t)^2 + d_{i-1}(r,s)^2 \Leftrightarrow d_{i-1}(s,t)}{2d_{i-1}(r,s)}\right)^2 \\
&= \frac{4d_{i-1}(r,s)^2 d_{i-1}(r,t)^2 \Leftrightarrow (d_{i-1}(r,t)^2 + d_{i-1}(r,s)^2 \Leftrightarrow d_{i-1}(s,t)^2)^2}{4d_{i-1}(r,s)^2} \\
&= \frac{1}{4d_{i-1}(r,s)^2}(4d_{i-1}(r,s)^2 d_{i-1}(r,t)^2 \Leftrightarrow d_{i-1}(r,s)^4 \\
&\qquad \Leftrightarrow d_{i-1}(r,s)^2(d_{i-1}(r,t)^2 \Leftrightarrow d_{i-1}(s,t)^2) \Leftrightarrow (d_{i-1}(r,t)^2 \Leftrightarrow d_{i-1}(s,t)^2)^2) \\
&= \frac{\Leftrightarrow d_{i-1}(r,s)^4 + 2d_{i-1}(r,s)^2(d_{i-1}(r,t)^2 + d_{i-1}(s,t)^2) \Leftrightarrow (d_{i-1}(r,t)^2 \Leftrightarrow d_{i-1}(s,t))^2}{4d_{i-1}(r,s)^2},
\end{aligned}$$

and

$$\begin{aligned}
d_i(s,t)^2 &= d_{i-1}(s,t)^2 \Leftrightarrow |x_s^{i-1} \Leftrightarrow x_t^{i-1}|^2 = d_{i-1}(s,t)^2 \Leftrightarrow |d_{i-1}(r,s) \Leftrightarrow x_t^{i-1}|^2 \\
&= d_{i-1}(s,t)^2 \Leftrightarrow \left(\frac{d_{i-1}(s,t)^2 + d_{i-1}(r,s)^2 \Leftrightarrow d_{i-1}(r,t)}{2d_{i-1}(r,s)}\right)^2 \\
&= \frac{4d_{i-1}(r,s)^2 d_{i-1}(s,t)^2 \Leftrightarrow (d_{i-1}(s,t)^2 + d_{i-1}(r,s)^2 \Leftrightarrow d_{i-1}(r,t)^2)^2}{4d_{i-1}(r,s)^2} \\
&= \frac{1}{4d_{i-1}(r,s)^2}(4d_{i-1}(r,s)^2 d_{i-1}(s,t)^2 \Leftrightarrow d_{i-1}(r,s)^4 \\
&\qquad \Leftrightarrow d_{i-1}(r,s)^2(d_{i-1}(s,t)^2 \Leftrightarrow d_{i-1}(r,t)^2) \Leftrightarrow (d_{i-1}(s,t)^2 \Leftrightarrow d_{i-1}(r,t)^2)^2) \\
&= \frac{\Leftrightarrow d_{i-1}(r,s)^4 + 2d_{i-1}(r,s)^2(d_{i-1}(s,t)^2 + d_{i-1}(r,t)^2) \Leftrightarrow (d_{i-1}(s,t)^2 \Leftrightarrow d_{i-1}(r,t))^2}{4d_{i-1}(r,s)^2}
\end{aligned}$$

Clearly, the final formulas for the two are the same, even if the triangle inequality is not satisfied. ■

Next, we give the conditions under which distances between objects are preserved.

Lemma 9 *Let r and s be some data objects, not necessarily pivot objects. If $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for all objects t after $i \Leftrightarrow 1$ iterations of FastMap, then $d'(F(r),F(s)) = d(r,s)$, assuming that Equation 9 is used to compute distances for each iteration. In other words, the distance between r and s is preserved by F.*

Proof Since $d_i(r,s) = 0$, we have $d(r,s) = d_E(F_{i-1}(r), F_{i-1}(s))$, where the notation $F_{i-1}(v)$ means the $(i \Leftrightarrow 1)$-dimensional partial vector for v determined in the first $i \Leftrightarrow 1$ iterations. Furthermore, since $d_i(r,t) = d_i(s,t)$ for all objects t, coordinate values i throught k will be the same for r and s. Thus, $d(r,s) = d_E(F(r),F(s))$, and since d' is the Euclidean distance metric d_E, we have $d(r,s) = d'(F(r),F(s))$ as desired. ■

From Lemmas 8 and 9, we can see that the distances between pivot objects is preserved. We can use the observation in Lemma 9 to define equivalence classes for the data objects, which in turn enables us to make a more general statement on conditions that ensure distance preservation

for a pair of objects. In particular, we say that two objects r and s (not necessarily pivot objects) belong to the same equivalence class in iteration i if $d_i(r,s) = 0$ and $d_i(r,t) = d_i(s,t)$ for all objects t. Initially, each object belongs to a distinct equivalence class (except for objects having a distance of zero from each other). Furthermore, each iteration of FastMap causes the merger of at least two equivalence classes, namely the ones containing the two pivot objects. Thus, at the conclusion of FastMap, the distances between all objects belonging to the same equivalence class are preserved, assuming that d' is the Euclidean distance metric. In other words, $d(r,s) = d'(F(r),F(s))$ if r and s belong to the same equivalence class.

The maximum number of equivalence classes after i iterations is clearly $N \Leftrightarrow i$, as at each iteration we merge at least two equivalence classes into one. This leads directly to the conclusion that the maximum possible number of iterations is $N \Leftrightarrow 1$, in which case the number of equivalence classes is one. This means that all pairwise distances between the N objects are preserved by the embedding — that is, $d(r,s) = d'(F(r),F(s))$ for all $\binom{N}{2}$ pairs of objects r and s. However, it is important to note that given N objects, it is not always possible to reduce the number of equivalence classes to one. In particular, this is the case when the intermediate mapping F_{i-1} is non-contractive after some number $i \Leftrightarrow 1$ of iteration. When this occurs, some negative distance values $d_i(r,s)^2$ exist, and because $d_j(r,s)^2$ will also be negative for $j > i$, this means that equivalence classes for r and s can never be merged. In other words, this shows that, in general, it is not possible to preserve all distances with FastMap, no matter how many iterations are performed. Nevertheless, the reasoning above is another proof that when d is the Euclidean metric, FastMap leads to a distance-preserving embedding for sufficiently large value of k, since in this case, the squared projected distances are never negative.

Until now, we have assumed that the heuristic described in Section 5.8 is not used. Lemma 8 can be shown to also hold when using Equation 10 to compute the intermediate distance function d_i ($i \geq 2$). However, in this case, the observation in Lemma 9 may no longer hold if d is not the Euclidean distance metric. In particular, the proof made use of the fact that $d(r,s) = d_E(F_{i-1}(r),F_{i-1}(s))$ when $d_i(r,s) = 0$. This property does not hold if any of the values $d_j(r,s)$, $2 \leq j < i$, is negative for the objects r and s in Lemma 9. Therefore, if d is not Euclidean, then distance preservation cannot be achieved whether or not the heuristic is used.

7 Concluding Remarks

We have examined a number of embeddings of finite metric spaces and evaluated them in the context of their usage for similarity searching in multimedia databases with 100% recall. 100% recall is important in similarity search as it ensures that no relevant object is dropped from the query result. Particular attention was paid to Lipschitz embeddings (as exemplified by SparseMap) and FastMap which is designed to be a heuristic alternative to the KLT (and the equivalent PCA and SVD) method for dimensionality reduction. 100% recall is achieved when the resulting mapping is contractive (i.e., the pruning property is satisfied). Although Linial, London and Rabinovich [19] showed how to make the Lipschitz embeddings contractive, we showed that the speedup heuristics that comprise the SparseMap adaptation make the resulting mapping non-contractive. Moreover, we showed how to modify the SparseMap heuristics so that the resulting mapping is indeed contractive.

In the case of FastMap, we first proved that it was contractive when the data is drawn from a Euclidean space, and the distance metric in the embedding space is a Minkowski metric L_p ($p \geq 2$ which also includes the Euclidean distance metric). Although in their development of FastMap, Faloutsos and Lin [5] claim that the advantage of FastMap over methods such as SVD is that

FastMap can work for data drawn from an arbitrary metric space (i.e., the only information about the data objects are 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 proved 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 [5] 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. Non-contractiveness 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 non-positive, 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 of FastMap.

A heuristic for non-Euclidean distance metric in FastMap was proposed in [23]. This heuristic 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 likelyhood 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.

References

[1] 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. *Journal of the ACM*, 45(6):891–923, November 1998.

[2] M. Bern. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 45(2):95–99, February 1993.

[3] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1–2):46–52, 1985.

[4] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.

[5] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD Conference*, pages 163–174, San Jose, CA, May 1995.

[6] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Boston, second edition, 1990.

[7] G. H. Golub and C. F. van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

[8] J. Hafner, H. Sawhney, W. Equitz, and M. Flickner et al. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):729–736, July 1995.

[9] G. R. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. under preparation.

[10] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Advances in Spatial Databases — Fourth International Symposium, SSD'95*, M. J. Egenhofer and J. R. Herring, eds., pages 83–95, Portland, ME, August 1995. (Also Springer-Verlag Lecture Notes in Computer Science 951).

[11] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, June 1999.

[12] G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins. Technical report, Rutgers University, 1999.

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

[14] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, (26):189–206, 1984.

[15] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 215–226, Mumbai, India, September 1996.

[16] J. B. Kruskal and M. Wish. Multidimensional scaling. Technical report, Sage University Series, Beverly Hills, 1978.

[17] M. Linial, N. Linial, N. Tishby, and G. Yona. Global self organization of all known protein sequences reveals inherent biological signatures. *Journal of Molecular Biology*, 268(2):539–556, May 1997.

[18] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic application s. In *Proceedings 35th IEEE Annual Symposium on Foundations of Computer Science*, pages 577–591, Santa Fe, NM, November 1994.

[19] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[20] K. W. Pettis, T. A. Bailey, A. K. Jain, and R. C. Dubes. An intrinsic dimensionality estimator from near-neighbor information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(1):25–37, 1979.

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

[22] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the ACM SIGMOD Conference*, pages 154–165, Seattle, WA, June 1998.

[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. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 307–311, San Diego, CA, August 1999.

[24] F. W. Young and R. M. Hamer. *Multidimensional scaling: History, theory, and applications.* Lawrence Erlbaum Associates, Hillsdsle, NJ, 1987.

[25] N. J. Young. *An introduction to Hilbert space.* Cambridge University Press, 1988.