JAIST Repository
https://dspace.jaist.ac.jp/

Title	A Linear-Space Algorithm for Distance Preserving Graph Embedding
Author(s)	Asano, Tetsuo; Bose, Prosenjit; Carmi, Paz; Maheshwari, Anil; Shu, Chang; Smid, Michiel; Wuhrer, Stefanie
Citation	Computational Geometry : Theory and Applications, 42(4): 289-304
Issue Date	2009-05
Type	Journal Article
Text version	author
URL	http://hdl.handle.net/10119/8809
Rights	NOTICE: This is the author's version of a work accepted for publication by Elsevier. Changes resulting from the publishing process, including peer review, editing, corrections, structural formatting and other quality control mechanisms, may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Tetsuo Asano, Prosenjit Bose, Paz Carmi, Anil Maheshwari, Chang Shu, Michiel Smid and Stefanie Wuhrer, Computational Geometry : Theory and Applications, 42(4), 2009, 289-304, http://dx.doi.org/10.1016/j.comgeo.2008.06.004
Description	

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

A Linear-Space Algorithm for Distance Preserving Graph Embedding*

Tetsuo Asano[1] Prosenjit Bose[2] Paz Carmi[2] Anil Maheshwari[2]
Chang Shu[3] Michiel Smid[2] Stefanie Wuhrer[2,3]

Abstract

The *distance preserving graph embedding problem* is to embed the vertices of a given weighted graph onto points in d-dimensional Euclidean space for a constant d such that for each edge the distance between their corresponding endpoints is as close to the weight of the edge as possible. If the given graph is complete, that is, if the weights are given as a full matrix, then multi-dimensional scaling [13] can minimize the sum of squared embedding errors in quadratic time. A serious disadvantage of this approach is its quadratic space requirement. In this paper we develop a linear-space algorithm for this problem for the case when the weight of any edge can be computed in constant time. A key idea is to partition a set of n objects into $O(\sqrt{n})$ disjoint subsets (clusters) of size $O(\sqrt{n})$ such that the minimum inter cluster distance is maximized among all possible such partitions. Experimental results are included comparing the performance of the newly developed approach to the performance of the well-established least-squares multi-dimensional scaling approach [13] using three different applications. Although least-squares multi-dimensional scaling gave slightly more accurate results than our newly developed approach, least-squares multi-dimensional scaling ran out of memory for data sets larger than 15000 vertices.

1 Introduction

Suppose a set of n objects is given and for each pair (i, j) of objects their dissimilarity denoted by $\delta_{i,j}$ can be computed in constant time. Using the dissimilarity information, we want to map the objects onto points in a low dimensional Euclidean space while preserving the dissimilarities as the distances between the corresponding points.

Converting distance information into coordinate information is helpful for human perception because we can see how close two objects are. Many applications require the embedding of a set of

*A preliminary version of this work appeared at CCCG 2007 [2]. Research supported in part by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas, Scientific Research (B), HPCVL, NSERC, NRC, MITACS, and MRI. We thank attendees of the 9th Korean Workshop on CG and Geometric Networks 2006. Work by T.A. was done in 2006 while visiting MPI, Carleton University, and NYU. We wish to thank anonymous referees for helpful comments.

[1]Japan Advanced Institute of Science and Technology, Ishikawa, Japan.
[2]Carleton University, Ottawa, Canada.
[3]National Research Council of Canada, Ottawa, Canada.

high dimensional objects while preserving dissimilarities for data analysis and visualization. Examples include analysis of psychological models, stock markets, computational chemistry problems, and image and shape recognition. Because of its practical importance, this topic has been widely studied under the name of dimensionality reduction [4].

Multi-Dimensional Scaling (MDS) [13] is a generic name for a family of algorithms for dimensionality reduction. Although MDS is powerful, it has a serious drawback for practical use due to its high space complexity. The input to MDS is an $n \times n$ matrix specifying the pairwise dissimilarities (or distances).

In this paper, we present a method for dimensionality reduction that avoids this high space complexity if the dissimilarity information is given by a function that can be evaluated in constant time. In many cases, we can assume that dissimilarity information is given by a function that can be evaluated in constant time. For instance, if we are given k-dimensional data vectors, where k is a large constant, we can compute the Euclidean distance between two data vectors in constant time. High dimensional data-vectors are ubiquitous in the field of computer vision, where the vectors represent images, and the field of machine learning, where the vectors represent characteristic dimensions of the data. An algorithm requiring linear space is preferable to an algorithm requiring quadratic space for large datasets, even if the algorithm requiring linear space is slower than the algorithm requiring quadratic space. The reason is that the time to load data from external memory dominates the running time of the algorithm in practice. Hence, the use of external memory should be avoided by using the linear-space algorithm.

Contribution

Given a set S of n objects and a function that computes the dissimilarity $\delta_{i,j}, 1 \leq i, j \leq n$ with $\delta_{i,j} = \delta_{j,i}$ between a pair of objects in $O(1)$ time, the objective is to find a set $P(S)$ of points $p_1, \ldots, p_n$ in d-dimensional space, such that $E_{LSMDS} = \sum_{1 \leq i < j \leq n} (d(p_i, p_j) - \delta_{i,j})^2$ is minimized, where $d(p_i, p_j)$ is the Euclidean distance between the embedded points p_i and p_j in $\mathbb{R}^d$. We aim to perform this computation using linear space.

A key idea of our linear-space implementation is to use clustering in the first step. That is, we partition the set S into $O(\sqrt{n})$ disjoint subsets called *clusters*. More precisely, using a positive integer m, we partition S into $k = O(\frac{n}{m})$ subsets $C_1, C_2, \ldots, C_k$ such that each cluster contains between m and cm objects, where $c \geq 2$ is a constant, except for possibly one cluster having at most m elements. When we set $m = O(\sqrt{n})$ then both the number, k, of clusters and the largest cluster size are bounded by $O(\sqrt{n})$. Hence, each cluster has size $O(\sqrt{n})$.

To compute the embedding, we first find a center for each cluster and embed the cluster centers using MDS. Since there are $O(\sqrt{n})$ clusters this takes $O(n)$ space. Once the cluster centers are embedded, we add the remaining objects by embedding one cluster at a time using MDS. Note that performing MDS with a distance matrix for each cluster separately requires only $O(n)$ working space.

The quality of the output depends heavily on the clustering. A clustering achieves the largest inter-cluster distance if the smallest distance between objects from different clusters is maximized. A clustering achieves the smallest inner-cluster distance if the largest distance between objects from the same cluster is minimized. The best clustering that can be achieved is a clustering that achieves the largest inter-cluster distance and the smallest inner-cluster distance simultaneously. Unfortunately, it is NP-hard to find a clustering that achieves the smallest inner-cluster distance. However, we provide an algorithm that in the special case of so-called "well-separated" partitions

as defined in Section 3.1 finds the clustering that achieves the two goals simultaneously.

Since it is in general not possible to find a clustering achieving both goals, we relax the clustering condition. In many applications, dissimilarities between similar objects have more importance than those between totally dissimilar objects. We propose a simple algorithm for finding a size-constrained clustering that achieves the largest inter-cluster distance.

Experimental results are included comparing the performance of the newly developed approach to the performance of the well-established least-squares multi-dimensional scaling approach [13]. We apply the embedding algorithms to three different application areas and show that our algorithm requires significantly less space than the least-squares multi-dimensional scaling approach, with comparable accuracy.

Organization

The paper is organized as follows. Section 2 reviews related work and gives an overview of applications of dimensionality reduction using only linear space. Section 3 presents our clustering algorithm that finds $O(\sqrt{n})$ clusters of size $O(\sqrt{n})$ each with largest inter-cluster distance. Section 4 presents the algorithm to embed the clusters in a low-dimensional space while minimizing the sum of squared embedding errors over all pairwise dissimilarities using only $O(n)$ space. Section 5 gives experimental results of our implementation. Finally, Section 6 concludes and gives ideas for future work.

2 Related Work

This section reviews work related to dimensionality reduction. We start by giving applications of dimensionality reduction using linear space.

2.1 Applications of Dimensionality Reduction

Problems where we wish to embed dissimilarities as points in a low dimensional space often arise in many different settings including data visualization [18], data classification [14], and computer graphics [9]. The goal is to find meaningful low-dimensional subspaces that are hidden in the high-dimensional observations. Low-dimensional subspaces are especially useful to visualize the high-dimensional data in a meaningful way.

Embedding high-dimensional datasets is a common task in data classification. We therefore demonstrate embedding a large dataset of registered high-energy gamma particles with our algorithm in Section 5. Section 5 further applies our algorithm to embed grey-level images of faces to a low-dimensional space. This embedding is useful in applications such as face recognition or expression recognition.

A problem arising in computer graphics is surface matching or recognition. Given a database of three-dimensional triangulated surfaces, we aim to efficiently find the instance in the database that is closest to a new triangulated model. An especially hard variant of this problem is matching deformable objects. Examples of deformable surfaces are human faces and bodies, since the muscle deformations and skeletal movements of the human yield new shapes of the human face and body. Matching human faces (this is also called 3D face recognition) or bodies requires defining similarity of triangulated deformable surfaces. In the case of the human body, we can model the deformations as

isometries. That is, geodesic distances on the surface are invariant during deformation. Clearly, this model is not completely accurate since human skin can stretch, but the assumption approximates the valid deformations and yields good matching algorithms [6, 7, 8, 9]. When matching isometric deformable surfaces, the pairwise geodesic distances between vertices on the triangular surface can be used as dissimilarities to perform MDS. Elad and Kimmel [15] observed that after embedding the vertices of the surface in a low dimensional space via MDS, we obtain a point cloud that is invariant with respect to deformations of the original surface. This point cloud is denoted as the *canonical form* of the original surface. Elad and Kimmel show via experimental results that performing the matching step after performing MDS yields accurate matching results. We use this application to demonstrate the use of our algorithm in Section 5.

2.2 Multi Dimensional Scaling

Multi-Dimensional Scaling (MDS) [13] is a general and powerful framework for constructing a configuration of points in a low-dimensional space using information on the pairwise dissimilarities $\delta_{i,j}$. Given a set S of n objects as well as the dissimilarities $\delta_{i,j}, 1 \leq i, j \leq n$ with $\delta_{i,j} = \delta_{j,i}$, the objective is to find a set $P(S)$ of points $p_1, \ldots, p_n$ in d-dimensional space, such that the Euclidean distance between p_i and p_j equals $\delta_{i,j}$. Since this objective can be shown to be too ambitious, we aim to find a good approximation. Different related optimality measures can be used to reach this goal.

Classical MDS, also called Principal Coordinate Analysis (PCO), assumes that the dissimilarities are Euclidean distances in a high-dimensional space and aims to minimize

$$E_{PCO} = \sum_{1 \leq i < j \leq n} |d(p_i, p_j)^2 - \delta_{i,j}^2|, \tag{1}$$

where $d(p_i, p_j)$ is the Euclidean distance between the embedded points p_i and p_j in $\mathbb{R}^d$. Equation (1) is minimized by computing the d largest eigenvalues and corresponding eigenvectors of a matrix derived from the distance matrix [13]. Thus, if we neglect some numerical computational issues concerning eigenvalue computation, the PCO embedding for n objects can be computed in $O(n^2 d)$ time using quadratic space. The reason is that only the d eigenvectors corresponding to the largest eigenvalues need to be computed. These can be computed in $O(n^2 d)$ time using a variation of the power method [22]. Note that PCO's result can be poor when the $(d+1)$-st largest eigenvalue is not negligible compared to the d-th largest one for embedding into d-space.

Least-squares MDS (LSMDS) aims to minimize

$$E_{LSMDS} = \sum_{1 \leq i < j \leq n} (d(p_i, p_j) - \delta_{i,j})^2. \tag{2}$$

Equation (2) can be minimized numerically by using scaling by maximizing a convex function (SMACOF) [13]. However, the algorithm can get stuck in local minima. The embedding can be computed in $O(n^2 t)$ time using quadratic space, where t is the number of iterations required by the SMACOF algorithm.

Although PCO and LSMDS are powerful techniques, they have serious drawbacks because of their high space complexity, since the input is an $n \times n$ matrix specifying pairwise dissimilarities (or distances).

The recent approach of anchored MDS [10] enhances MDS by allowing to embed groups of objects in the following way. The data is interactively divided into two groups, a group of objects called

anchors, and a group of objects called floaters. The anchors either have fixed coordinates or are embedded using MDS. The floaters are then embedded with respect to the anchors only. That is, anchors affect the embedding of floaters, but floaters do not affect the embedding of anchors. This approach is conceptually similar to the clustering approach taken in this paper. However, no clustering is performed in anchored MDS; the groups are given to the algorithm as input.

2.3 Other Dimensionality Reduction Algorithms

Two other popular algorithms for dimensionality reduction are locally linear embedding (LLE) [24] and isomap [25]. LLE is similar to our algorithm in that the points are clustered and clusters are later recombined. However, the clustering used in LLE is a heuristic and cannot be proven to satisfy the property that the minimum distance between objects in any two different clusters is maximized. We will show later that the clustering presented in this paper has this desirable property. Furthermore, the recombination step of LLE requires quadratic working space. Isomap embedding uses classical MDS as a subroutine and therefore uses quadratic working space. The advantage of using an isomap embedding is that the algorithm can be proven to converge to the global optimal embedding.

Another class of dimensionality reduction algorithms aims to analytically bound the worst embedding error [11, 12].

3 Clustering

In this section, we consider the problem of partitioning a set of objects into clusters according to the values of the pairwise dissimilarities. We show that it is possible to find a size-constrained partitioning with largest inter-cluster distance using linear space.

Let S be a set of n objects: $S = \{1, \ldots, n\}$. Assume that we are given a function which computes the *dissimilarity* between any pair (i, j) of objects as $\delta_{i,j}$ with $\delta_{i,i} = 0$ and $\delta_{i,j} = \delta_{j,i} > 0$ for $i \neq j$. A partition $\mathcal{P}$ of a set S into k disjoint clusters $C_1, \ldots, C_k$ is called a *k-partition of S*. A k-partition $\mathcal{P}$ is characterized by two distances, *inner-cluster distance* $D_{inn}(\mathcal{P})$ and *inter-cluster distance* $D_{int}(\mathcal{P})$, which are defined as

$$D_{inn}(\mathcal{P}) = \max_{C_i \in \mathcal{P}} \max_{p,q \in C_i} \delta_{p,q}, \tag{3}$$

and

$$D_{int}(\mathcal{P}) = \min_{C_i \neq C_j \in \mathcal{P}} \min_{p \in C_i, q \in C_j} \delta_{p,q}. \tag{4}$$

When we define a complete graph $G(S)$ with edge weights being dissimilarities, edges are classified as inner-cluster edges connecting vertices of the same cluster and inter-cluster edges between different clusters. The inner-cluster distance is the largest weight of any inner-cluster edge and the inter-cluster distance is the smallest weight of any inter-cluster edge.

A k-partition is called *farthest* (*most compact*, respectively) if it is a k-partition with largest inter-cluster distance (smallest inner-cluster distance, respectively) among all k-partitions. Given a set S of n objects, we want to find a k-partition of S which is farthest and most compact. It is generally hard to achieve the two goals simultaneously. In fact, the problem of finding a most compact k-partition, even in the special case where the dissimilarities come from a metric space, is NP-hard [16]. There are, however, cases where we can find such a k-partition rather easily. One easy case is the case of a well-separated partition.

3.1 Well-separated partitions

A k-partition $\mathcal{P}$ of a set S is called well-separated if $D_{inn}(\mathcal{P}) < D_{int}(\mathcal{P})$. We prove that if there is a well-separated k-partition of a given set then we can find a k-partition that is farthest and most compact. Moreover the partition is unique if no two pairs of objects have the same dissimilarity.

Lemma 3.1 *Let S be a set of n objects with a dissimilarity defined for every pair of objects. If S has a well-separated k-partition, then it is unique provided that no two pairs of objects have the same dissimilarity. The unique well-separated k-partition is farthest and most compact.*

Proof: The uniqueness of a well-separated k-partition follows from the assumption that no two pairs of objects have the same dissimilarity. A k-partition classifies edges of a complete graph $G(S)$ defined by dissimilarities into inner-cluster edges and inter-cluster edges. Then, the inner-cluster distance $D_{inn}(\mathcal{P})$ is achieved by the largest inner-cluster edge and the inter-cluster distance $D_{int}(\mathcal{P})$ by the smallest inter-cluster edge. Since $\mathcal{P}$ is well-separated, this means that for every inner-cluster edge, its weight is smaller than that of every inter-cluster edge. So, if we change any edge from an inner-cluster to an inter-cluster edge or vice versa then we violate the inequality $D_{inn}(\mathcal{P}) < D_{int}(\mathcal{P})$. Thus, uniqueness follows.

Let $\mathcal{P}$ be the unique well-separated k-partition of S. Then, it is most compact since reducing the inner-cluster distance requires splitting some clusters, which increases the number of clusters. It is also farthest. To prove it by contradiction, suppose that there is a k-partition (which is not necessarily well separated) with inter-cluster distance $t > D_{int}(\mathcal{P})$. This means that all the edges which were inner-cluster edges remain unchanged and those edges with weight greater than $D_{inn}(\mathcal{P})$ and less than t have become inner-cluster edges. So, the number of connected components must be at most $k-1$, which is a contradiction. □

Consider the case where S admits a well-separated k-partition. By Lemma 3.1, if we sort all the dissimilarities in increasing order then the inter-cluster distance must appear right next to the inner-cluster distance in this order. So, it suffices to find some dissimilarity t^* such that there is a well-separated k-partition with the inner-cluster distance being t^*. If we define a graph $G_t(S)$ as the subgraph of $G(S)$ with all the edges of weight at most t then the connected components of $G_t(S)$ define a well-separated partition of S with inner-cluster distance t. So, if we can find a dissimilarity t such that the graph $G_t(S)$ consists of k connected components, then it is a solution. The following is an algorithm for counting the number of connected components in a graph $G_t(S)$ using linear working space. Each cluster is stored in a linked list. In the algorithm, we first scan every pair and if its weight is at most t then we merge the two clusters containing those objects into one cluster. At this moment we report the number of remaining clusters. After that, we again scan every pair and report NO if we find a pair with dissimilarity greater than t such that both of them belong to the same cluster, and report YES if no such pair is found.

If S has a well-separated k-partition, the algorithm returns k and YES for dissimilarity $\delta_{i,j}$ for some pair (i, j). A naive algorithm is to check all the dissimilarities. Since there are $O(n^2)$ different dissimilarities, $O(n^2)$ iterations of the algorithm are enough. It takes $O(n^4)$ time in total but the space required is $O(n)$.

An idea for efficient implementation is binary search on the sorted list of dissimilarities. Generally speaking, the larger the t value becomes the fewer subsets we have by the algorithm above. If the output is k and YES for some t^*, then the resulting partition is the unique well-separated k-partition we want. If there is such a value t^*, any other value of t with $t > t^*$ generates fewer clusters. On the other hand, if we have more than k clusters for some t, then we have $t < t^*$. There can be many t

values that generate exactly k clusters, but t^* is the largest value among them. Thus, if the output is NO and the number of clusters is at most k for some t then we can conclude that $t < t^*$. Based on this observation, we can implement a binary search.

Linear-space algorithm for well-separated partition: One serious problem with the method sketched above is that we cannot store a sorted list of dissimilarities due to the linear space constraint. We implement the binary search in two stages. In the beginning, our search interval contains a superlinear number of distances. So, we compute an approximate median instead of the exact median. As the binary search proceeds, our interval gets shorter and shorter. Once the number of distances falling into the search interval is at most cn for some positive constant c, then we can find an exact median. A more detailed description follows:

We start our binary search from the initial interval $[1, \binom{n}{2}]$ which corresponds to a distance interval determined by the smallest and largest distances. We maintain an index interval $[low, high]$ corresponding to the distance interval $[\delta_{low}, \delta_{high}]$, where δ_i denotes the i-th smallest distance. Imagine dividing the interval $[low, high]$ into 4 equal parts, then an approximate median is any element contained in the 2nd or 3rd quarters. Thus, half of the elements in $[low, high]$ are approximate medians. Equivalently, a random element is an approximate median with probability 1/2. How can we find one?

We pick a random integer k with $1 \leq k \leq high - low + 1$. We can evaluate the dissimilarity function in the order in which the dissimilarities are encountered when scanning the (unknown) distance matrix row by row to simulate scanning the distance matrix. We refer to this process, which uses only $O(1)$ space, as *scanning the matrix*. We scan the matrix row by row and pick the k-th element X with $\delta_{low} \leq X \leq \delta_{high}$ that we encounter. Given X, we scan the matrix and count the number of values between δ_{low} and X, and also count the number of values between X and δ_{high}. In this way, we find out if X is an approximate median. If it is not, then we repeat the above. We know that the expected number of trials is 2. Assume that X is a good approximate median. While doing the above, we also find the index m such that $X = \delta_m$. Now we test if X is equal/larger/smaller than D_{inn}. If they are equal, we are done. Assume X is less than D_{inn}. Then, we set the right boundary $high$ of our current interval to m. If X is larger than D_{inn}, then we set the left boundary low to m.

In this way, we spend $O(n^2)$ time for one binary search step. Since the expected number of these steps is $O(\log n)$, the overall expected time bound is $O(n^2 \log n)$. Once the current interval contains at most cn distances, we can apply an exact median finding algorithm although we have to scan the matrix in $O(n^2)$ time.

Theorem 3.2 *Given n objects, a function evaluating the dissimilarity between any pair of objects in $O(1)$ time, and an integer $k < n$, we can decide whether or not there is a well-separated k-partition in $O(n^2 \log n)$ expected time and $O(n)$ working space using approximate median finding. Moreover, if there is such a partition, we find it in the same time and space.*

If we are allowed to use $O(n \log n)$ space, then we can further improve the running time using another randomization technique as follows. For ease of notation, assume that the dissimilarities are numbers $1, 2, 3, \ldots, n^2$. Define the interval $I_i = [(i - \varepsilon/2)n, (i + \varepsilon/2)n]$ for $i = 1, 2, \ldots, n$. We would like to obtain one element from each of these intervals. We choose a random sample consisting of $cn \log n$ elements. What is the probability that this sample leaves one of the intervals I_i empty? This probability is at most

the probability that we did not choose any element of I_1
+ the probability that we did not choose any element of $I_2 + \cdots$
+ the probability that we did not choose any element of I_n.

All probabilities above are equal, so let us look at the probability that we did not choose any element of I_1. The probability that a random element is not in I_1 equal to $1-(\varepsilon n)/n^2 = 1-\varepsilon/n \leq e^{-\varepsilon/n}$. So the probability that none of the $cn \log n$ random elements is in I_1 is at most $(e^{-\varepsilon/n})^{cn \log n}$, which is $n^{-c'}$ for some constant c'. Thus the total probability that our random sample is not good is at most $n \times n^{-c'}$, which is at most $1/2$. In other words, the probability that the random sample is good is at least $1/2$. This means that we expect to do the random sampling at most twice. Hence, the expected running time of the algorithm is $O(n^2)$.

3.2 Farthest and Most Compact k-partition

The problem of finding a most compact k-partition with the smallest inner-cluster distance is difficult since even in the special case that the dissimilarities come from a metric space, finding the most compact k-partition for $k \geq 3$ is NP-hard [16, 19]. However, we can find a farthest k-partition rather easily.

Let $e_1, e_2, \ldots, e_{n-1}$ be the edges of a minimum spanning tree $MST(S)$ for a complete graph $G(S)$ defined for a set S of n objects, and assume that

$$|e_1| \leq |e_2| \leq \cdots \leq |e_{n-1}|. \tag{5}$$

Let $MST_k(S)$ be the set of components resulting after removing the $k-1$ longest edges $e_{n-1}, \ldots, e_{n-k+1}$ from $MST(S)$. Then, $MST_k(S)$ has exactly k components, which defines a k-partition of S. The following lemma has been claimed by Asano et al. [1] and is proven in Kleinberg and Tardos [21, p. 160].

Lemma 3.3 *Given a set S of n objects with all pairwise dissimilarities defined, there is an algorithm for finding a farthest k-partition in $O(n^2)$ time and linear working space.*

3.3 Size-constrained Farthest partition

Recall that our aim is to embed the graph using only $O(n)$ space. In order to use MDS for the embedding, clusters that are farthest are not sufficient. We also need to ensure that the clusters are sufficiently small and that there are not too many of them. Specifically, we need to find $O(\sqrt{n})$ clusters of size $O(\sqrt{n})$ each. For a given m with $1 < m < n$, define a *farthest partition satisfying the size constraint on m* as a clustering $\mathcal{P}$ where all clusters contain between m and cm vertices, where $c \geq 2$ is a constant, at most one cluster contains at most m vertices, and $D_{int}(\mathcal{P})$ is maximized. The method outlined in Lemma 3.3 does not provide such a partitioning.

To find the farthest partition satisfying the size constraint on m given a set S of n objects, consider the following algorithm. First, each object i is placed into a separate cluster C_i of size one to initialize the algorithm. The algorithm iteratively finds the minimum remaining dissimilarity $\delta_{i,j}$. If merging the cluster C_l containing object i and cluster C_q containing object j does not violate the size constraint, that is, if it does not produce a new cluster of size exceeding cm, the algorithm merges C_l and C_q into one cluster C_l. Dissimilarity $\delta_{i,j}$ is then removed from consideration. These steps are iterated until all of the dissimilarities are removed from consideration. We denote this algorithm Size-constrained-farthest-partition(m) in the following.

Lemma 3.4 *Given m with $1 < m < n$ and a constant $c \geq 2$, algorithm Size-constrained-farthest-partition(m) creates a partition such that all clusters contain between m and cm vertices except for at most one cluster that contains at most m vertices.*

Proof: None of the clusters created by this algorithm has size greater than cm, since otherwise, the clusters would not have been merged by the algorithm. It remains to prove that there exists at most one cluster of size at most m. Assume for the sake of a contradiction that there exist two clusters C_l and C_q of size at most m. Since C_l and C_q have not yet been merged, all $\delta_{i,j}$ with $i \in C_l$ and $j \in C_q$ have not yet been removed from consideration. Hence, the algorithm has not yet reached its stopping criterion, which contradicts the assumption. Hence, the clusters C_l and C_q are merged before the termination of the algorithm and therefore, there exists at most one cluster that contains at most m vertices. □

Note that the algorithm created k clusters with $\lfloor \frac{n}{cm} \rfloor < k \leq \lceil \frac{n}{m} \rceil + 1$, because all clusters contain at most cm vertices and at most one cluster contains at most m vertices according to the pigeon hole principle.

Lemma 3.5 *The partition created by algorithm Size-constrained-farthest-partition(m) is farthest among all partitions with the property that all clusters contain at most cm vertices and at most one cluster contains at most m vertices, where $c \geq 2$ is a constant.*

Proof: Assume for the sake of a contradiction that the partition $\mathcal{P}$ created by the algorithm described above is not farthest. Hence, there exists a farther optimal partition $\mathcal{P}_{OPT}$ with the property that all clusters contain at most cm vertices and at most one cluster contains at most m vertices. The partition $\mathcal{P}$ has $D_{int}(\mathcal{P}) = \delta_{i,j}$ with $i \in D_l$ and $j \in D_q$, where D_l and D_q are the first clusters that do not get merged by our algorithm. Since $\mathcal{P}_{OPT}$ is farther, $\delta_{i,j}$ is an internal edge of $\mathcal{P}_{OPT}$. Since all clusters of $\mathcal{P}_{OPT}$ contain at most cm vertices, it is impossible for all vertices of D_l and D_q to belong to one cluster of $\mathcal{P}_{OPT}$. Hence, without loss of generality, there exists a vertex a in D_l that is in a different cluster of $\mathcal{P}_{OPT}$ than i. Next, we will show that there exists a path p_{ai} between a and i inside D_l comprised entirely of edges of length at most $\delta_{i,j}$. Initially, i and a were located in two different components C_i and C_a in the execution of the algorithm described above. Since during its execution, the algorithm merged the component C_a containing a and the component C_i containing i before considering the edge $\delta_{i,j}$, there exists an element $b \in C_a$ and an element $c \in C_i$ with $\delta_{b,c} \leq \delta_{i,j}$. This argument can be applied recursively to the pairs of vertices a, b and i, c, respectively, until the path p_{ai}, comprised entirely of edges of length at most $\delta_{i,j}$, is found. For an illustration of this approach, please refer to Figure 1.

Since in $\mathcal{P}_{OPT}$, a is in a different cluster than i, at least one of the edges on the path p_{ai} is an inter-cluster edge. Hence, the inter-cluster distance $D_{int}(\mathcal{P}_{OPT}) \leq D_{int}(\mathcal{P})$, which contradicts the initial assumption. This proves that the partition $\mathcal{P}$ is farthest. □

To find this partition, we need to find the shortest edge that has not yet been considered iteratively until all the edges are considered. In algorithm Size-constrained-farthest-partition(m) we use a data structure for extracting edges in increasing order of their weights.

To analyze the running time of algorithm Size-constrained-farthest-partition(m), note that finding and storing the minimum dissimilarity of each row of the matrix takes $O(n^2)$ time. There are at most $O(n^2)$ iterations to find the smallest dissimilarity that has not yet been considered. One iteration takes $O(n)$ time, since we store and update the minimum dissimilarity of each row of the matrix. Hence, the total running time is $O(n^3)$.

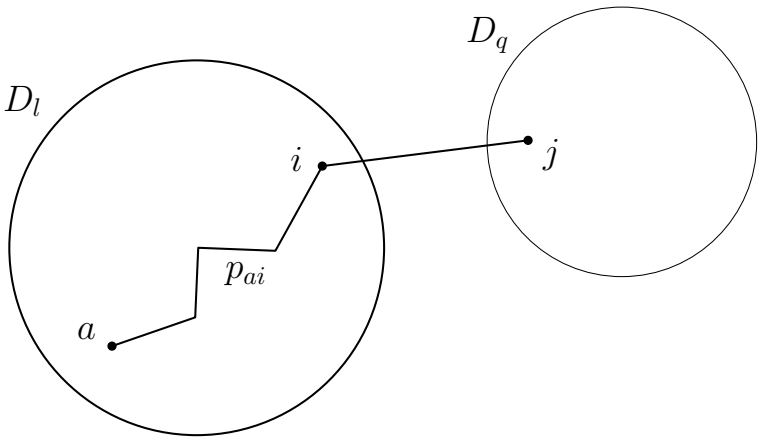


Figure 1: Path p_{ai} comprised of short edges between i and a.

Theorem 3.6 *One can compute the farthest partition satisfying the size constraint on m in $O(n^3)$ time using $O(n)$ space.*

An example of running the clustering algorithm on a set of points in the plane is shown in Figure 2. In this example, the dissimilarity is defined to be the Euclidean distance between a pair of points. Clusters are shown as connected sets. Cluster centers are shown as black points.

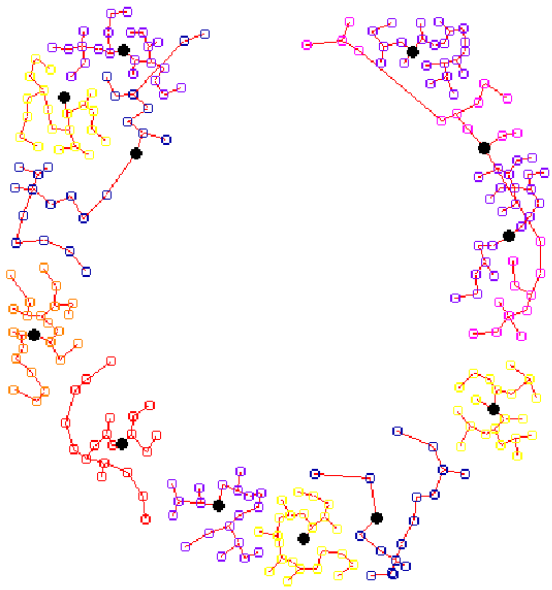


Figure 2: Example showing the result of running the clustering algorithm on a random set of 268 points in the plane with $m = 2$ and $c = 11$.

4 Graph Embedding

A direct way of embedding a weighted graph into a low-dimensional space is to apply least-squares multi-dimensional scaling (LSMDS), see Section 2.2, which needs a full matrix representing dissim-

ilarities between all pairs of objects. This takes $O(n^2)$ space for a set of n objects, which is often a problem for implementation when n is large. To remedy this, we partition the given set into $O(\sqrt{n})$ clusters of size $O(\sqrt{n})$ each by applying Algorithm Size-constrained-farthest-partition(m) with $m = \sqrt{n}$. Consider the $k = O(\sqrt{n})$ clusters $C_1, C_2, \ldots, C_k$ with $n_i = |C_i| = O(\sqrt{n})$ for each i.

First we find a center object in each cluster C_i, denoted by $center(C_i)$, which is defined to be an object in C_i whose greatest dissimilarity with any other object in the cluster is the smallest. We denote the i-th cluster by $C_i = \{p_1^i, p_2^i, \ldots, p_{n_i-1}^i\}$. For ease of notation, we exclude the cluster center $p_i = p_{n_i}^i$ from the cluster C_i in the following.

Second, we form a set $C_0 = \{p_1, p_2, \ldots, p_k\}$ consisting of cluster centers. Since $k = O(\sqrt{n})$, we can apply LSMDS to find an embedding that minimizes the sum of squared embedding errors of elements in C_0 using a distance matrix of size $O(n)$. We fix those points as embedded positions $X_0 = [\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_k]$.

Third, we embed clusters $C_1, C_2, \ldots, C_k$, $k = O(\sqrt{n})$, one by one. We do this by minimizing a generalized form of the least-squares MDS energy function given in Equation (2). The energy that is minimized to embed cluster $C_i, i = 1, 2, \ldots, k$ is

$$E = \underbrace{\sum_{p_j \in C_i} \sum_{p_k \in C_i} (\delta_{j,k} - d(p_j, p_k))^2}_{E_1} + \underbrace{\sum_{p_j \in C_i} \sum_{p_k \in C_0} (\delta_{j,k} - d(p_j, p_k))^2}_{E_2}. \tag{6}$$

Denote the position vector of the embedding of object $p_j \in C_i$ in $\mathbb{R}^d$ by $\vec{x}_j = [x_{j,1}\ x_{j,2}\ \ldots\ x_{j,d}]^T$ and denote the point matrix by $X_i = \left[\vec{x}_{i_1}\ \vec{x}_{i_2}\ \ldots\ \vec{x}_{i_{n_i-1}}\right]^T$.

The first part of the energy function, E_1, corresponds to the complete LSMDS energy if only points in the same cluster C_i are considered. The energy E_1 can therefore be expressed as

$$E_1 = \alpha + \beta - \gamma$$

with $\alpha = \sum_{p_j \in C_i} \sum_{p_k \in C_i} \delta_{j,k}^2, \beta = \sum_{p_j \in C_i} \sum_{p_k \in C_i} d(p_j, p_k)^2$ and $\gamma = 2 \sum_{p_j \in C_i} \sum_{p_k \in C_i} d(p_j, p_k)\delta_{j,k}$ and using the Cauchy-Schwartz inequality bounded by

$$E_1 \leq \sum_{p_j \in C_i} \sum_{p_k \in C_i} \delta_{i,j}^2 + tr(X_i^T V_i X_i) - 2tr(X_i^T B_i(Z_i) Z_i) =: \tau^*,$$

where X_i is a $d \times (n_i - 1)$ matrix containing the coordinates of p_i, V_i is an $(n_i - 1) \times (n_i - 1)$ matrix with elements

$$V_{i_{j,k}} = \begin{cases} n_i - 1 \text{ if } j = k \\ -1 \text{ if } j \neq k, \end{cases}$$

Z_i is a possible solution for X_i, and $B_i(Z_i)$ is an $(n_i - 1) \times (n_i - 1)$ matrix with elements

$$B_{i_{j,k}} = \begin{cases} -\frac{\delta_{j,k}}{d(p_j^i, p_k^i)} \text{ if } j \neq k, d(p_j^i, p_k^i) \neq 0 \\ 0 \text{ if } j \neq k, d(p_j^i, p_k^i) = 0 \\ \sum_{l=1, l \neq j}^{n} B_{l,j} \text{ if } j = k. \end{cases}$$

The gradient of τ^* can be found analytically as $\nabla \tau^* = 2X_i^T(V_i - B_i)$ [5, p.146-155].

The second part of the energy, E_2, considers the distances between the cluster being embedded and the fixed cluster centers. It is the same energy used when adding the points in C_i to the fixed embedding of C_0 one by one. The energy can be rewritten as

$$E_2 = \sum_{p_j \in C_i} \alpha_j^* + \beta_j^* - \gamma_j^*$$

where $\alpha_j^* = \sum_{p_k \in C_0} \delta_{j,k}^2$, $\beta_j^* = \sum_{p_k \in C_0} d(p_j, p_k)^2$ and $\gamma_j^* = 2 \sum_{p_k \in C_0} \delta_{j,k} d(p_j, p_k)$. The gradient of E_2 can therefore be computed analytically as

$$\nabla E_2 = \left[\frac{\partial E_2}{\partial p_1^i} \; \frac{\partial E_2}{\partial p_2^i} \; \cdots \; \frac{\partial E_2}{\partial p_{n_i-1}^i} \right],$$

where $\frac{\partial E_2}{\partial p_k^i} = \sum_{\vec{x}_j \in X_0} 2(\vec{x}_{i_k} - \vec{x}_j)(1 - \frac{\delta_{j,k}}{d(p_j, p_k^i)})$ [3].

Hence, we can compute the gradient of the convex function $\tau = \tau^* + E_2$ bounding E from above with respect to X_i analytically as

$$\nabla \tau = 2X_i^T(V_i - B_i) + \left[\frac{\partial E_2}{\partial p_1^i} \; \frac{\partial E_2}{\partial p_2^i} \; \cdots \; \frac{\partial E_2}{\partial p_{n_i-1}^i} \right]. \tag{7}$$

We minimize τ using a quasi-Newton method called *limited-memory Broyden-Fletcher-Goldfarb-Shanno* (LSBFGS) scheme in our experiments [23]. This quasi-Newton method offers the advantage of obtaining close to quadratic convergence rates without the need to compute the inverse of the Hessian matrix explicitly in each step. Instead LSBFGS updates an approximation to the inverse Hessian matrix in each iterative step, such that the approximation converges to the true inverse of the Hessian in the limit. To be consistent, when adding clusters to the embedding, we initialize the embedding positions to the positions computed when adding an object to the PCO embedding of the cluster centers [17].

Since we aim to minimize τ, we embed each cluster optimally according to the LSMDS optimality measure when taking into consideration the cluster centers and the objects in the same cluster. We do not take points other than the cluster centers from different clusters into account, since this would require more than linear space. This can be viewed as a trade-off between space and accuracy of the embedding.

The running time of this algorithm, referred to as the *embedding algorithm* in the following, depends on the maximum number of iterations t. Embedding the cluster centers takes $O(nt)$ time and embedding each cluster C_i takes $O(nt)$ time. Hence, the total running time of the embedding algorithm is $O(n^{\frac{3}{2}}t)$. Since all of the matrices required for the computation are of size at most $O(\sqrt{n}) \times O(\sqrt{n})$, the algorithm uses linear working space.

Recall that the algorithm to compute a size-constrained farthest partition takes $O(n^3)$ time. This yields the following result:

Theorem 4.1 *Given a set S of n objects, the embedding algorithm embeds S in $\mathbb{R}^d$ for any constant embedding dimension d while numerically minimizing the sum of squared embedding errors of pairwise dissimilarities between objects in S using $O(n^3 + n^{\frac{3}{2}}t)$ time and $O(n)$ space, where t is the number of iterations required to minimize the LSMDS energy.*

5 Experiments

We implemented our algorithm of Theorem 4.1 in C++. All of the experiments were conducted on an Intel (R) Pentium (R) D with 3.5 GB of RAM. We compare our algorithm to two alternative embedding algorithms: the SMACOF algorithm explained in Cox and Cox [13] to compute an LSMDS embedding using quadratic space and a variant of our algorithm that embeds the cluster centers using SMACOF and simply adds the remaining objects one by one to the embedding. The two algorithms are explained below.

5.1 SMACOF Algorithm

The SMACOF algorithm aims to minimize the objective function E_{LSMDS} given in Equation (2). Since E_{LSMDS} is hard to minimize, it is easier to iteratively approximate the objective function by a simple function. This approach is used in the algorithm *Scaling by Maximizing a Convex Function* (SMACOF) that is explained by Borg and Groenen [5, p.146-155] and used by Elad and Kimmel [15] to compute canonical forms. SMACOF proceeds by iteratively refining a simple majorization function that bounds the objective function E_{LSMDS} from above.

As in Section 4, we rewrite the objective function as

$$E_{LSMDS} = \alpha + \beta - \gamma$$

with $\alpha = \sum_{i=1}^{n}\sum_{j=i+1}^{n}\delta_{i,j}^2$, $\beta = \sum_{i=1}^{n}\sum_{j=i+1}^{n} d(p_i,p_j)^2$ and $\gamma = 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} d(p_i,p_j)\delta_{i,j}$ and using the Cauchy-Schwartz inequality bound it by

$$E_{LSMDS} \leq \sum_{i=1}^{n}\sum_{j=i+1}^{n}\delta_{i,j}^2 + tr(X^TVX) - 2tr(X^TB(Z)Z) =: \tau^*,$$

where X is a $d \times n$ matrix containing the coordinates of p_i, V is an $n \times n$ matrix with elements

$$V_{i,j} = \begin{cases} n-1 \text{ if } i = j \\ -1 \text{ if } i \neq j, \end{cases}$$

Z is a possible solution for X, and $B(Z)$ is an $n \times n$ matrix with elements

$$B_{i,j} = \begin{cases} -\frac{\delta_{i,j}}{d(p_i,p_j)} \text{ if } i \neq j, d(p_i,p_j) \neq 0 \\ 0 \text{ if } i \neq j, d(p_i,p_j) = 0 \\ \sum_{l=1,l\neq i}^{n} B_{i,l} \text{ if } i = j. \end{cases}$$

We implement the SMACOF algorithm and minimize τ^* using a quasi-Newton method. The quasi-Newton method used is the *limited-memory Broyden-Fletcher-Goldfarb-Shanno* (LSBFGS) scheme [23]. The running time of the SMACOF algorithm is $O(n^2t)$, where t is the number of iterations used by the algorithm.

5.2 Projection Algorithm

We can obtain an approximate graph embedding using a variant of our proposed embedding algorithm as follows. We first compute the farthest k-partition with $k = \lfloor\sqrt{n}\rfloor$ as presented in

Section 3.2. The cluster centers are objects whose greatest dissimilarity with any other object in the cluster is smallest. We first embed the cluster centers using the SMACOF algorithm and then add all of the remaining objects to the LSMDS embedding one by one [3]. This approach takes only linear space, but it does not take any inner-cluster distances into consideration when embedding the objects. For ease of notation, we call this algorithm the *projection algorithm* in the following discussion.

We describe how to add an additional object O_{n+1} with corresponding dissimilarities $\delta_{n+1,1}, \ldots, \delta_{n+1,n}$ that becomes available only after the objects $O_1, O_2, \ldots, O_n$ have been mapped to points $p_1, p_2, \ldots, p_n$ in d-dimensional space [3]. We minimize the least-squares function

$$E^*_{LSMDS} = \sum_{i=1}^{n} \left(\delta_{n+1,i} - d(p_{n+1}, p_i)\right)^2,$$

which can be written as

$$E^*_{LSMDS} = \alpha^* + \beta^* - \gamma^*$$

where $\alpha^* = \sum_{i=1}^{n} \delta^2_{n+1,i}$, $\beta^* = \sum_{i=1}^{n} d(p_{n+1}, p_i)^2$ and $\gamma^* = 2\sum_{i=1}^{n} \delta_{n+1,i} d(p_{n+1}, p_i)$.

We can now compute the gradient of this objective function w.r.t. the point p_{n+1} analytically as

$$\nabla E^*_{LSMDS} = \sum_{i=1}^{n} 2(\vec{x}^T_{n+1} - \vec{x}^T_i)\left(1 - \frac{\delta_{n+1,i}}{d(p_{n+1}, p_i)}\right),$$

where $\vec{x}_i$ is the column vector of coordinates of point p_i. This allows us to add the object O_{n+1} to the MDS embedding by minimizing E^*_{LSMDS} using an LSBFGS quasi-Newton approach [23].

5.3 Comparison

For all embeddings that are computed, we initialize the embedding to the PCO embedding and use the LSBFGS quasi-Newton method for the iterative minimization. For all of our experiments, the constant c for the clustering step is chosen as 2 and the parameter m is chosen as $\lfloor\sqrt{n}\rfloor$. In the following, three applications are considered.

We compare our embedding algorithm to the SMACOF algorithm and the projection algorithm in terms of storage requirement, produced embedding error, running time, and maximum number of iterations t to minimize the energy using LSBFGS. The storage requirement of all algorithms ignores the space to store that data. In case of computing canonical forms, the comparison therefore ignores the space allocated to store the triangular mesh and the data structures required to compute geodesic distances along the mesh. To measure the amount of storage used by the algorithms, we recursively report all the memory that is allocated.

The experiments show that the time complexity of the distance function has a big influence on the running time of the algorithms. It is therefore recommended to evaluate the distance function as efficiently as possible.

5.3.1 Embedding Registered High-Energy Gamma Particles

In a first experiment, we embed the MAGIC gamma telescope data available in the UCI Machine Learning Repository[1] into $\mathbb{R}^3$. The data was generated using a Monte-Carlo method and simulates

[1] http://archive.ics.uci.edu/ml/

the registration of high-energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope. The dataset consists of 10-dimensional vectors of real numbers. The distance between two data vectors is computed as Euclidean distance. Note that this implies a low time complexity of the distance function, since the Euclidean distance between 10-dimensional vectors can be computed efficiently. There are a total of 19020 data vectors available in the dataset. The algorithms were tested on five different sizes of datasets. We obtain five datasets of size 1000 to 19020 simply by taking a subset of the original dataset.

Table 1 compares the performance of the three algorithms on different sizes of data of the gamma telescope dataset. The table shows the quality of the computed embeddings, the size and space complexities of the algorithms, and the number of iterations required by the algorithms. We do not give the results of the SMACOF algorithm for the two largest datasets, since the algorithm ran out of memory. The time and space requirements are furthermore visualized in Figure 3.

The storage required by the projection algorithm and the embedding algorithm are significantly smaller than the storage required by the SMACOF algorithm. This is to be expected, since the SMACOF algorithm requires $O(n^2)$ storage while the other two algorithms require $O(n)$ storage. In this experiment, we see an example where SMACOF is no longer practical due to its quadratic space complexity. For $n = 10000$, we can no longer initialize the embedding to the one computed using classical MDS since there is not enough internal memory. This leads to an abnormality in the running time and embedding quality of the SMACOF result for $n = 10000$. For $n \geq 15000$, we can no longer run the SMACOF algorithm in internal memory. In order to compute a SMACOF embedding for the datasets of size 15000 or larger, we need to store data in external memory, thereby significantly growing the time complexity of the algorithm. In contrast, the data stored by the embedding algorithm and the projection algorithm still easily fits into internal memory. The projection algorithm requires only about half of the storage required by the embedding algorithm and is the most space efficient algorithm of the three.

The embedding algorithm is the most time efficient algorithm for this dataset. The reason is that the time of evaluating the input dissimilarity function is small compared to the matrix operations used by SMACOF. In general, SMACOF is expected to be the most time efficient algorithm, since its running time is $O(tn^2)$, while the running time of the other two algorithms is $O(n^3)$. The quality of the computed embedding is highest for SMACOF. This does not hold for $n = 10000$, since it was no longer possible to initialize the embedding to the classical MDS embedding in this case. We expect SMACOF to be most accurate because SMACOF takes all of the pairwise dissimilarities into account when computing the embedding. The second most accurate embedding is the one produced by the embedding algorithm. This is also expected as the embedding algorithm takes inner-cluster dissimilarities into account while the projection algorithm does not.

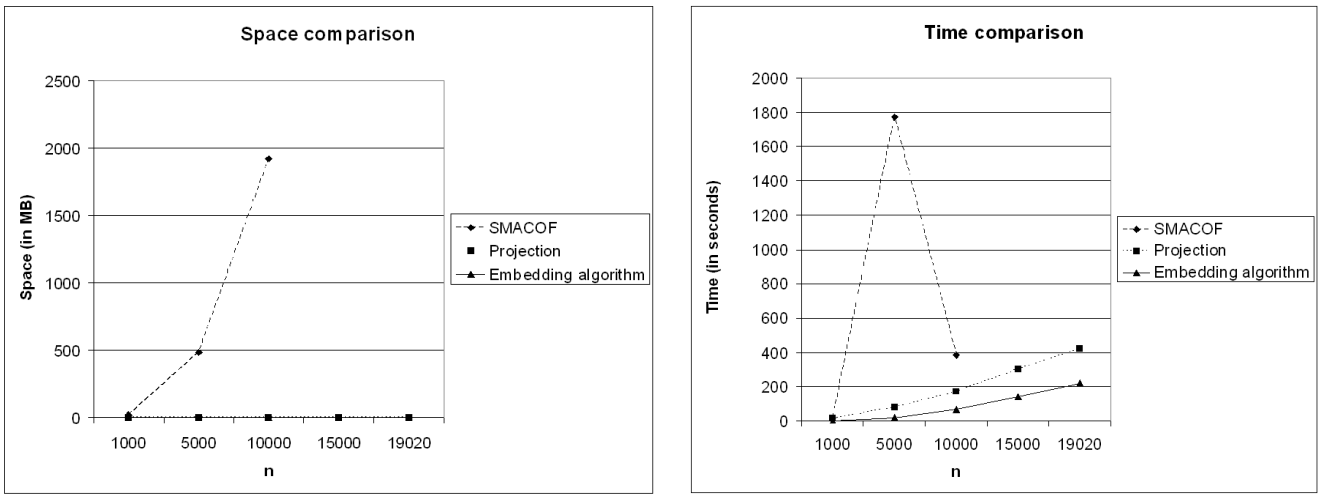
5.3.2 Embedding Grey-Level Images

The second application is embedding grey-level images of faces into points in the plane. We use the Yale Face Database[2] showing the faces of 15 individuals. The face of each individual is shown in 11 different expressions. The eleven facial expressions of one of the subjects in the database are shown in Figure 4. An embedding of these faces can be used for classification by subject or expression or for face recognition. Each grey-level image is a vector of dimension 77760. The distance between two grey-level images is computed as the Euclidean vector-distance. Note that the running time of evaluating this distance function is significantly larger than the running time

[2]http://cvc.yale.edu/projects/yalefaces/yalefaces.html

n	Emb. alg.				Proj. alg.				SMACOF			
	t	E_{LSMDS}	S	time	t	E_{LSMDS}	S	time	t	E_{LSMDS}	S	time
1000	149	$5.3 \cdot 10^7$	0.19	1	43	$6.5 \cdot 10^7$	0.10	14	90	$2.6 \cdot 10^7$	20.28	17
5000	209	$1.3 \cdot 10^9$	0.58	18	77	$2.2 \cdot 10^9$	0.25	77	117	$7.0 \cdot 10^8$	482.73	1772
10000	198	$4.6 \cdot 10^9$	1.04	67	48	$9.8 \cdot 10^9$	0.42	173	52	$7.0 \cdot 10^{10}$	1919.02	384
15000	219	$1.8 \cdot 10^{10}$	1.48	141	59	$3.5 \cdot 10^{11}$	0.58	301	not enough internal memory			
19020	182	$3.6 \cdot 10^{10}$	1.82	221	72	$1.9 \cdot 10^{12}$	0.70	422	not enough internal memory			

Table 1: *Quality of embedding for the gamma telescope dataset. The table shows the maximum number t of iterations required by the LSBFGS quasi-Newton method [23], the embedding error E_{LSMDS} of the computed embedding, the storage use S in MB, and the running time in seconds of all three algorithms that were implemented.*



(a) (b)

Figure 3: *Comparison of time and space used by the embedding algorithm (solid), the projection algorithm (dotted), and the SMACOF algorithm (dashed) for the gamma telescope dataset. The number of objects n is shown along the x-axis and the space and time requirements are shown along the y-axis.*

of evaluating the distance function in the previous experiment because of the higher dimensionality of the data vectors.

The results are shown in Table 2. We can see that the space used by the embedding algorithm and the projection algorithm is significantly lower than the space used by SMACOF. The space used by the projection algorithm is about 70% of the space used by the embedding algorithm, since no inner-cluster distances need to be stored. The quality of all three embeddings is poor because the faces cannot be represented well in the plane. As expected, the result by SMACOF is best, followed by the result by the embedding algorithm. The SMACOF algorithm is fastest and the embedding algorithm is slowest. This is different than in the previous experiment. The reason is that it takes longer to compute the dissimilarity between two grey-level images than to compute the dissimilarities between two gamma particles.

Figure 4: *Facial expressions in the database.*

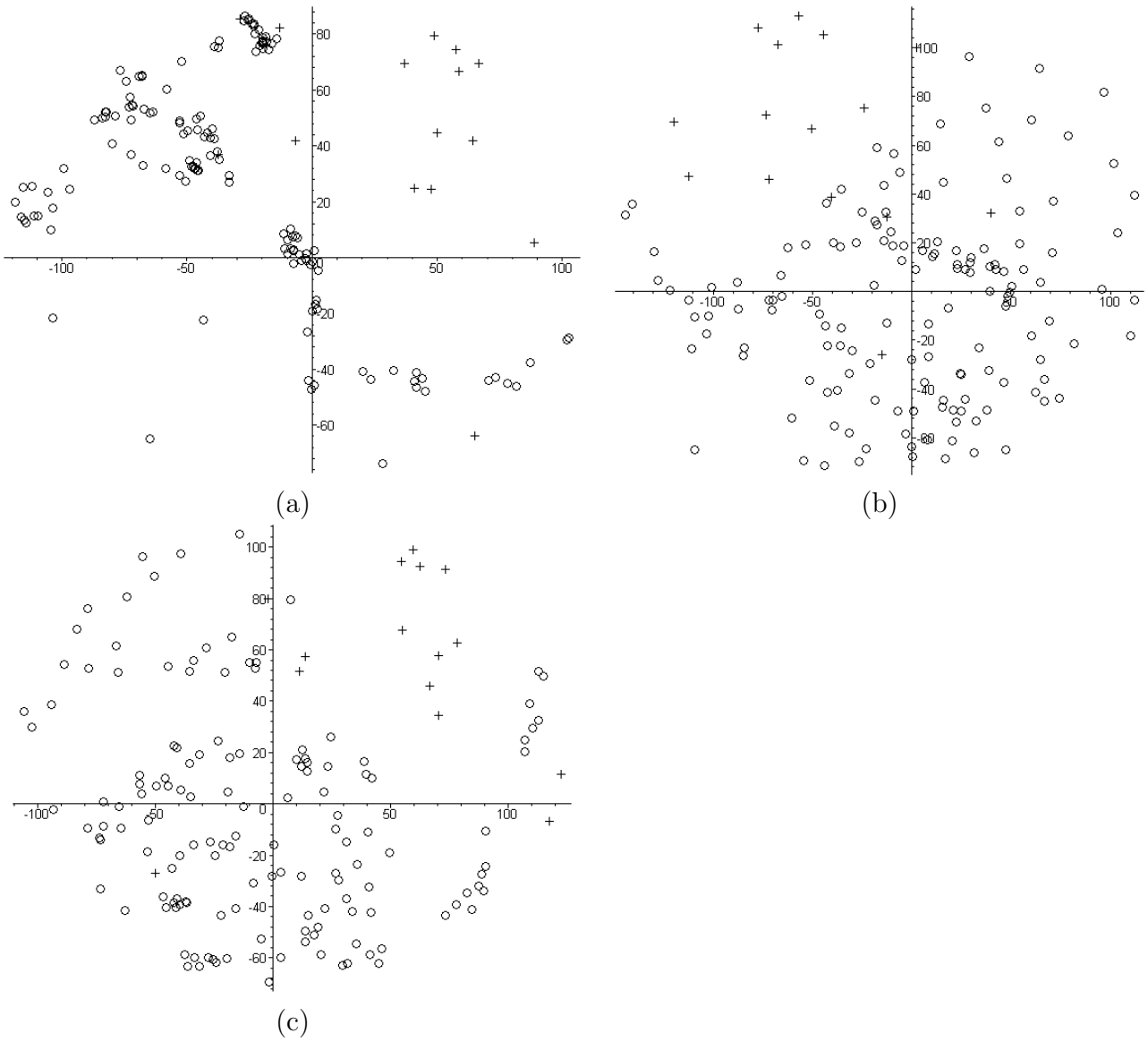


Figure 5: *The Figure shows the embedding results for the face database. Figure (a): embedding computed by the embedding algorithm. Figure (b): embedding computed by the projection algorithm. Figure (c): embedding computed by SMACOF.*

n	165			
	t	E_{LSMDS}	S (MB)	time (sec)
Emb. alg.	174	$1.17 \cdot 10^7$	0.070	84
Proj. alg.	48	$1.49 \cdot 10^7$	0.052	43
SMACOF	143	$8.63 \cdot 10^6$	0.69	9

Table 2: *Quality of embedding for the Yale Face Database.*

The embeddings are shown in Figure 5(a) to (c). Figure 5(a) shows the embedding obtained using the projection algorithm, Figure 5(b) shows the result obtained using the embedding algorithm, and Figure 5(c) shows the result obtained using the SMACOF algorithm. The points marked as crosses correspond to the expression where the face is lit from the right side only shown on the bottom left of Figure 4. We can see that most of the crosses are close in all of the embedding results. This suggests that we can visualize different expressions as clusters of points in the plane.

5.3.3 Computing a Canonical Form

The third application we consider is to compute the canonical form of a complete triangular surface [15]. In this application, objects are vertices on a triangular surface and dissimilarities between objects are geodesic distances between the corresponding vertices. Note that in this application, the assumption that the dissimilarity function can be evaluated in constant time does not hold, since it takes $O(n \log n)$ time to compute a geodesic distance [20]. Hence, the running time of the embedding algorithm becomes $O(n^4 \log n + n^{\frac{5}{2}} t \log n)$.

We evaluate the algorithms using the following two experiments. First, we evaluate the algorithms by embedding the surface of the swiss roll dataset into $\mathbb{R}^2$. The swiss roll dataset shown in Figure 8 (a) has a non-Euclidean structure, but can be rolled into a planar patch. Due to the complexity of unrolling the swiss roll, this experiment is commonly used to demonstrate the quality of embedding algorithms [24, 25, 9]. In our experiment, we use the parametric form of the swiss roll surface given by Bronstein et al. [9]: $x = \theta, y = 0.51(\frac{1}{2.75\pi} + 0.75\phi)\cos(2.5\phi), z = 0.51(\frac{1}{2.75\pi} + 0.75\phi)\sin(2.5\phi)$, where $(\theta, \phi) \in [0, 1] \times [0, 1]$.

Second, we evaluate the algorithms using the triangular mesh from the Princeton Shape Benchmark[3] shown in Figure 9(a) consisting of $n = 429$ vertices. We refine the mesh using local subdivision of triangles and run the algorithm on different resolutions of the mesh.

In the first (respectively second) experiment, we use geodesic distances on the mesh as dissimilarities and embed the vertices into $\mathbb{R}^2$ (respectively $\mathbb{R}^3$). Table 3 (respectively Table 4) reports the amount of storage required by the algorithms, the maximum number t of iterations required by the algorithms, the running time of the algorithms, and the error of the embedding computed by the algorithms according to equation (2). The time and space complexities of the algorithms are furthermore visualized in Figure 6 (respectively Figure 7). The x-axis of the graph shows the number n of vertices and the y-axes show the amount of storage used by the algorithms in MB and the running times of the algorithms in seconds.

We can see that the amount of storage required by the SMACOF algorithm shown as dashed curve grows significantly faster than the amount of storage required by the embedding algorithm shown as solid curve and the projection algorithm shown as dotted curve. This is to be expected, since the SMACOF algorithm takes $O(n^2)$ storage while the other two algorithms take $O(n)$ storage.

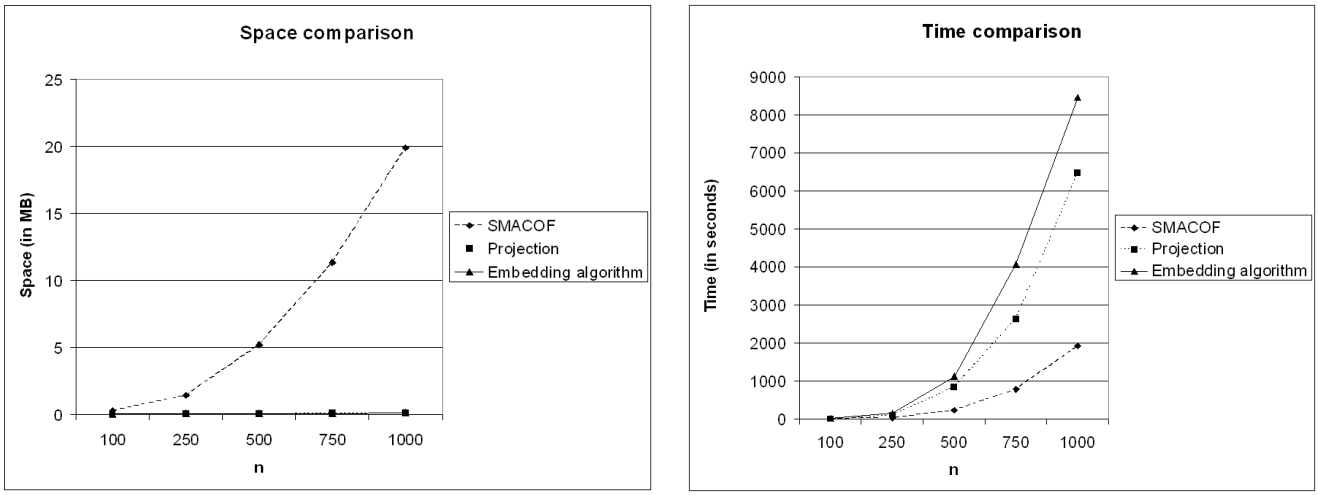
[3] http://shape.cs.princeton.edu/benchmark/

Furthermore, the projection algorithm requires only about half of the amount of storage required by the embedding algorithm.

The SMACOF algorithm is fastest and the embedding algorithm is slowest. Note that the difference in running time between the three algorithms is larger than in the previous experiment. The reason is that evaluating the dissimilarity function is slow in this experiment as the running time of the distance function is $O(n \log n)$.

n	Emb. alg.				Proj. alg.				SMACOF			
	t	E_{LSMDS}	S	time	t	E_{LSMDS}	S	time	t	E_{LSMDS}	S	time
100	1	$4.7 \cdot 10^{-27}$	0.062	12	1	$1.4 \cdot 10^{-27}$	0.049	8	1	$6.8 \cdot 10^{-28}$	0.31	2
250	1	$1.7 \cdot 10^{-26}$	0.081	149	1	$5.0 \cdot 10^{-26}$	0.056	108	1	$1.7 \cdot 10^{-27}$	1.42	26
500	1	$2.7 \cdot 10^{-025}$	0.11	1120	1	$1.5 \cdot 10^{-025}$	0.068	832	1	$1.0 \cdot 10^{-25}$	5.20	243
750	1	$5.3 \cdot 10^{-025}$	0.11	4070	1	$7.2 \cdot 10^{-025}$	0.077	2633	1	$1.0 \cdot 10^{-24}$	11.36	793
1000	1	$9.6 \cdot 10^{-24}$	0.17	8472	1	$1.2 \cdot 10^{-23}$	0.086	6462	1	$9.7 \cdot 10^{-24}$	19.90	1933

Table 3: *Quality of embedding for the swiss roll. The table shows the maximum number t of iterations required by the LSBFGS quasi-Newton method [23], the embedding error E_{LSMDS} of the computed embedding, the storage use S in MB, and the running time in seconds of all three algorithms that were implemented.*



(a) (b)

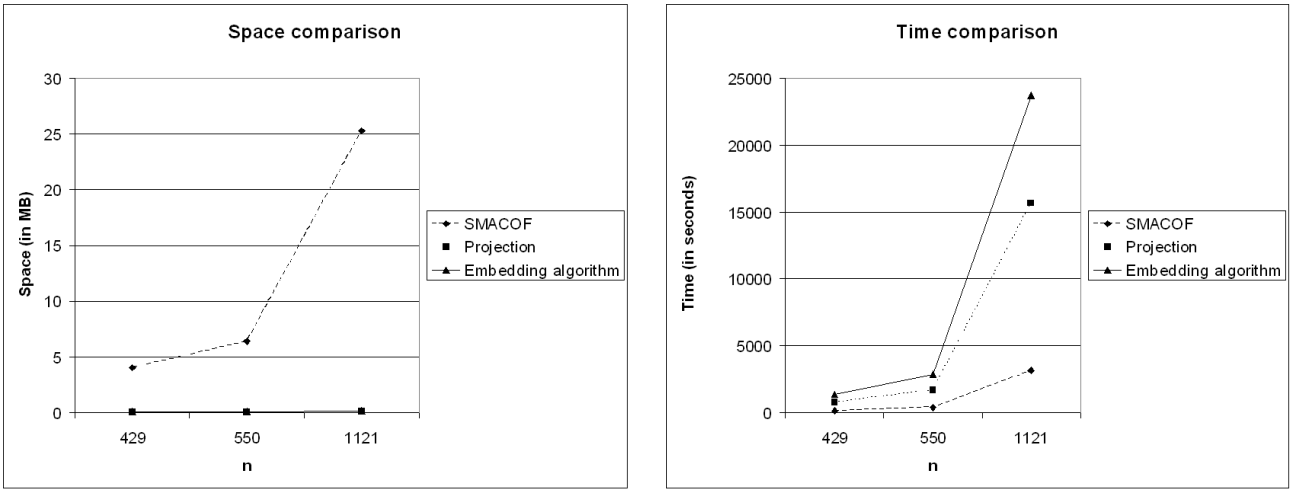
Figure 6: *Comparison of time and space used by the embedding algorithm (solid), the projection algorithm (dotted), and the SMACOF algorithm (dashed) for the swiss roll. The number of objects n is shown along the x-axis and the space and time requirements are shown along the y-axis.*

An image of the swiss roll containing 500 vertices as well as the embeddings computed using the tested algorithms are shown in Figure 8. Note that all of the results are similar except for rotations.

For the Alien dataset ($n = 429$), the embedding obtained using the projection algorithm is shown in Figure 9(b), the embedding obtained using the embedding algorithm is shown in Figure 9(c), and the embedding obtained using the SMACOF algorithm is shown in Figure 9(d). We can see that all of the embeddings are similar.

n	Emb. alg.				Proj. alg.				SMACOF			
	t	E_{LSMDS}	S	time	t	E_{LSMDS}	S	time	t	E_{LSMDS}	S	time
429	111	190	0.12	1403	91	233	0.073	807	99	950	4.05	162
550	143	170	0.13	2894	96	263	0.080	1680	152	126	6.45	374
1121	143	673	0.20	23720	93	1550	0.11	15663	128	540	25.32	3150

Table 4: *Quality of embedding for the Alien. The table shows the maximum number t of iterations required by the LSBFGS quasi-Newton method [23], the embedding error E_{LSMDS} of the computed embedding, and the storage use S of all three algorithms that were implemented.*



(a) (b)

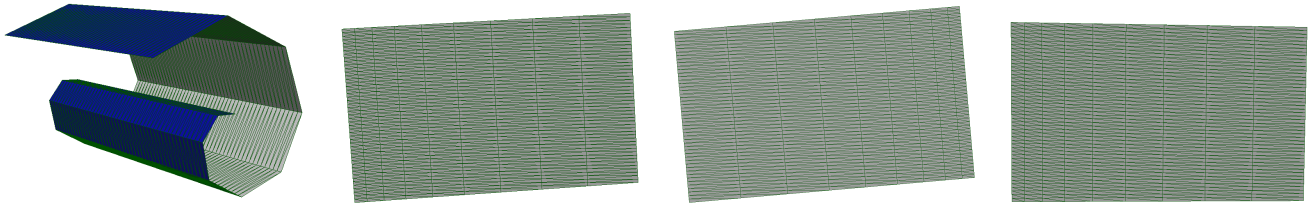
Figure 7: *Comparison of time and space used by the embedding algorithm (solid), the projection algorithm (dotted), and the SMACOF algorithm (dashed) for the Alien. The number of objects n is shown along the x-axis and the space and time requirements are shown along the y-axis.*

5.3.4 Summary

We conclude that both the embedding algorithm and the projection algorithm can compute an embedding of lower quality than the SMACOF embedding using significantly less storage. In most of our experiments, the error of the embedding computed using the embedding algorithm is about twice the error of the embedding computed using SMACOF. As expected, the embedding algorithm yields an embedding of higher quality than the projection algorithm.

Due to its linear space requirement, the embedding algorithm can be applied to compute embeddings for large datasets where storing a full dissimilarity matrix is no longer feasible as shown using the gamma telescope dataset. The embedding algorithm is especially useful to compute a low-dimensional embedding of a large dataset where the dissimilarity function can be evaluated fast. In this case, the embedding algorithm was shown to outperform the SMACOF algorithm in terms of running time. If the evaluation of the dissimilarity function is a large constant or a function whose running time grows as a function of n, then the embedding algorithm is slower than the SMACOF algorithm.

In all of our experiments, non-optimized code was used. The running time of the embedding algorithm possibly can be improved by embedding multiple clusters in parallel.



(a) (b) (c) (d)

Figure 8: *The Figure shows the swiss roll and its embeddings. Figure (a): swiss roll with* $n = 500$ *vertices. Figure (b): embedding computed by the embedding algorithm. Figure (c): embedding computed by the projection algorithm. Figure (d): embedding computed by SMACOF.*

(a) (b) (c) (d)

Figure 9: *Alien model used for evaluation of results. Figure (a) shows the original model, Figure (b) shows the embedding using the projection algorithm, Figure (c) shows the embedding using the embedding algorithm, and Figure (d) shows the embedding using SMACOF algorithm.*

6 Conclusions and Future Work

In this paper we considered the problem of embedding the vertices of a given weighted graph onto points in d-dimensional Euclidean space for a constant d such that for each edge the distance between their corresponding endpoints is as close as possible to the weight of the edge. We considered the case that the given graph is complete. Although MDS or PCO are powerful techniques for this purpose, they require quadratic space.

In this paper, we proposed a linear-space algorithm assuming that the dissimilarity for any pair of objects can be computed in constant time. Our experiments show that the quality of the embedding computed using the proposed linear-space algorithm is only lower by a factor of two than the quality of the embedding computed using the SMACOF algorithm. However, the space requirement of the embedding algorithm was shown to be significantly lower than the space requirement of the SMACOF algorithm.

An important open question is to find an algorithm to compute an embedding in a low-dimensional space that has provably low distortion using linear space.

References

[1] Tetsuo Asano, Binay Bhattacharya, Mark Keil, and Frances Yao. Clustering algorithms based on minimum and maximum spanning trees. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 252–257, 1988.

[2] Tetsuo Asano, Prosenjit Bose, Paz Carmi, Anil Maheshwari, Chang Shu, Michiel Smid, and Stefanie Wuhrer. Linear-space algorithm for distance preserving graph embedding with applications. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 185–188, 2007.

[3] Zouhour Ben Azouz, Prosenjit Bose, Chang Shu, and Stefanie Wuhrer. Approximations of geodesic distances for incomplete triangular manifolds. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 177–180, 2007.

[4] Holger Bast. Dimension reduction: A powerful principle for automatically finding concepts in unstructured data. In *In Proc. International Workshop on Self-Properties in Complex Information Systems (SELF-STAR 2004)*, pages 113–116, 2004.

[5] Ingwer Borg and Patrick Groenen. *Modern Multidimensional Scaling Theory and Applications*. Springer, 1997.

[6] Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Expression-invariant 3d face recognition. *Proceedings of the Audio- and Video-based Biometric Person Authentication, Lecture Notes in Computer Science*, 2688:62–69, 2003.

[7] Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Three-dimensional face recognition. *International Journal of Computer Vision*, 64(1):5–30, 2005.

[8] Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Robust expression-invariant face recognition from partially missing data. In *Proceedings of the European Conference on Computer Vision*, pages 396–408, 2006.

[9] Alexander M. Bronstein, Michael M. Bronstein, Ron Kimmel, and Irad Yavneh. Multigrid multidimensional scaling. *Numerical Linear Algebra with Applications, Special issue on multigrid methods*, 13(2–3):149–171, 2006.

[10] Andreas Buja, Deborah Swayne, Michael Littman, Nate Dean, Heike Hofmann, and Lisha Chen. Interactive data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, page to appear, 2008.

[11] Mihai Bădoiu, Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Piotr Indyk. Low-dimensional embedding with extra information. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 320–329, 2004.

[12] Mihai Bădoiu, Kedar Dhamdhere, Anupam Gupta, Yuri Rabinovich, Harald Räcke, R. Ravi, and Anastasios Sidiropoulos. Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 119–128, 2005.

[13] Trevor Cox and Michael Cox. *Multidimensional Scaling, Second Edition.* Chapman & Hall CRC, 2001.

[14] Richard Duda, Peter Hart, and David Stork. *Pattern Classification, Second Edition.* John Wiley & Sons, Inc., 2001.

[15] Asi Elad and Ron Kimmel. On bending invariant signatures for surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1285–1295, 2003.

[16] Teofilo Gonzalez. Clustering to minimize the maximum inter cluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[17] John C. Gower. Adding a point to vector diagrams in multivariate analysis. *Biometrika*, 55(3):582–585, 1968.

[18] Patrick Groenen and Philip H. Franses. Visualizing time-varying correlations across stock markets. *Journal of Empirical Finance*, 7:155–172, 2000.

[19] Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.

[20] Ron Kimmel and James Sethian. Computing geodesic paths on manifolds. *National Academy of Sciences*, 95:8431–8435, 1998.

[21] Jon Kleinberg and Eva Tardos. *Algorithm Design.* Addison-Wesley, 2005.

[22] Joseph Kruskal and Myron Wish. *Multidimensional Scaling.* Sage, 1978.

[23] Dong C. Liu and Jorge Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[24] Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 190(5500):2323–2326, 2000.

[25] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 190(5500):2319–2323, 2000.