p-ISOMAP: An Efficient Parametric Update for ISOMAP for Visual Analytics[*]

Jaegul Choo[†], Chandan K. Reddy[‡], Hanseung Lee[§],and Haesun Park[†]

Abstract

One of the most widely-used nonlinear data embedding methods is ISOMAP. Based on a manifold learning framework, ISOMAP has a parameter k or ϵ that controls how many edges a neighborhood graph has. However, a suitable parameter value is often difficult to determine because of a time-consuming optimization process based on certain criteria, which may not be clearly justified. When ISOMAP is used to visualize data, users might want to test different parameter values in order to gain various insights about data, but such interaction between humans and such visualizations requires reasonably efficient updating, even for large-scale data. To tackle these problems, we propose an efficient updating algorithm for ISOMAP with parameter changes, called p-ISOMAP. We present not only a complexity analysis but also an empirical running time comparison, which show the advantage of p-ISOMAP. We also show interesting visualization applications of p-ISOMAP and demonstrate how to discover various characteristics of data through visualizations using different parameter values.

1 Motivation

One of the most widely-used data mining techniques that reduce noise in data and improve efficiency in terms of computation time and memory usage is dimension reduction. Recently, nonlinear dimension reduction techniques, which have been actively investigated, revealed the underlying nonlinear structure in data. Such nonlinearity is often considered as a curvilinear manifold with a much lower dimension than that in the original high-dimensional space. Among the most recent nonlinear dimension reduction methods, isometric feature mapping (ISOMAP) has shown its effectiveness in capturing the underlying manifold structure in the reduced dimensional space by being successfully applied to synthetic data such as “Swiss roll” data and real-world data such as facial image data [16].

ISOMAP shares the basic idea with a traditional technique, classical multidimensional scaling (MDS). Classical MDS first constructs the pairwise similarity matrix, which is usually measured by the Euclidean distance, and computes the reduced dimensional mapping that maximally preserves such a similarity matrix in a given reduced dimension. ISOMAP differs from classical MDS in that it constructs the pairwise similarity matrix based on the geodesic distance estimated by the shortest path in the neighborhood graph of data. The neighborhood graph is formed by having vertices as data points and setting each edge weight between the nodes as their Euclidean distance only if at least one node is one of the k-nearest neighbors (k-NN) of the other node (k-ISOMAP) or if their Euclidean distance is smaller than ϵ (ϵ-ISOMAP). Hence, ISOMAP has an either parameter k or ϵ to construct the neighborhood graph. This paper focuses on the algorithm and applications of the dynamic updating of ISOMAP when the value of k or ϵ varies.

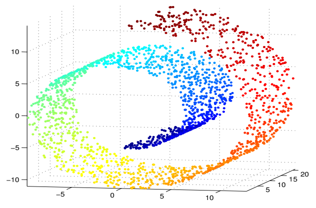
It is generally known that in ISOMAP, if k or ϵ is too small, the graph becomes sparse, resulting in infinite geodesic distances between some pairs of data points, and if k or ϵ is too large, it is prone to “short circuit” the true geometry of the manifold. However, it is not always easy to figure out which value of k or ϵ is appropriate for the data at hand. One way of optimizing these parameters is using certain quantitative measures such as residual variance [2, 16, 15] and finding the “elbow” point at which the residual variance curve stops decreasing significantly as the parameter value changes. However, running ISOMAP repeatedly using different parameter values for k or ϵ may be time-consuming since it involves computationally intensive processes such as the all-pairs shortest path computation and the eigendecomposition, whose complexity is usually $O(n^3)$

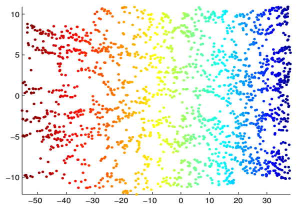
[*]The work of these authors was supported in part by the National Science Foundation grants DMS-0736328 and CCF-0808863. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

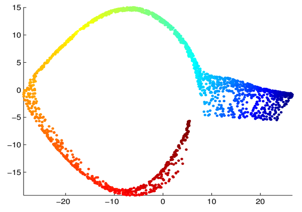
[†]School of Computational Science and Engineering, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA 30332, USA, {jaegul.choo, hpark}@cc.gatech.edu

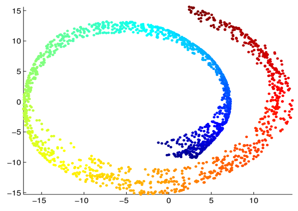
[‡]Dept. of Computer Science, Wayne State University, Detroit, MI 48202, USA, reddy@cs.wayne.edu

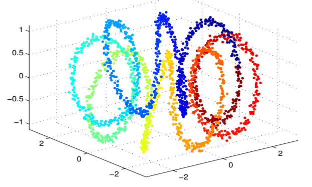
[§]School of ECE, Georgia Institute of Technology, 777 Atlantic Drive, Atlanta, GA 30332, USA, hanseung.lee@gatech.edu

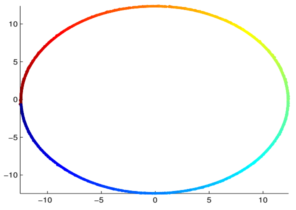

(a) The “Swiss roll” data set

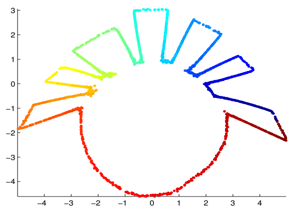

(b) ISOMAP with $k = 8$


(c) ISOMAP with $k = 40$


(d) ISOMAP with $k = 100$


(e) The toroidal helix data set


(f) ISOMAP with $k = 8$


(g) ISOMAP with $k = 40$

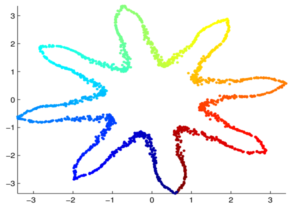

(h) ISOMAP with $k = 100$

Figure 1: ISOMAP examples with different k values. The first and second rows of figures correspond to the “Swiss roll” and the toroidal data sets, respectively.

in which n is the number of data points.[1]

Also in practice, there is often no guarantee of the existence of the underlying well-defined manifold structure in data, and thus, one may not be sure if manifold learning methods such as ISOMAP are suitable for the data at hand. Even so, one may still want to try ISOMAP or another manifold learning method in order to see if it serves one's purpose. In this case, however, it may not be a good idea to rely on a particular value of k or ϵ to achieve a reasonable dimension reduction since the optimal value tends to be indistinct in terms of a certain measure.

When it comes to the visualization of high-dimensional data in two- or three-dimensional space, we can acquire different insights on the data by using various dimension reduction techniques [6]. This statement also holds true even when we use just a single dimension reduction method, e.g., ISOMAP, while we test its various parameter values. In short, visualizations using ISOMAP with different parameter values for k or ϵ can provide us with various aspects of our data. In instances of the “Swiss roll” and toroidal helix data sets shown in Figure 1, one may want to visualize them based on the unfolded version of its manifold, as shown in Figures 1(b) and 1(f), but sometimes one may also want to see how the underlying manifold is curved in the original space, i.e., the curvature of the manifold itself, as shown in Figures 1(d) and 1(h).[2] It is also possible that visualizations of the transition between these two cases, shown in Figures 1(c) and 1(g), imply different insight about data. In this sense, it is worthwhile for users to test different parameter values in ISOMAP to visualize data in various ways. Such visualizations, however, should provide users with smooth and prompt interaction that requires fast and efficient computations of the results. In other words, when users change the parameter value, if they have to wait for a significant amount of time, then such interaction would not be practical.

Motivated by the above mentioned cases, we propose p-ISOMAP, an efficient dynamic updating algorithm for ISOMAP when the parameter value changes. Given the ISOMAP result from a particular parameter value, our proposed algorithm updates the previous result to obtain another ISOMAP result of the same data with a new parameter value instead of recomputing ISOMAP for different parameter values from scratch. We present the complexity analysis of our algorithms as well as the experimental comparison of their computation times. In addition, we demonstrate several visualization examples by varying the parameters in ISOMAP, which not only show the interesting aspects of the tested data but also help us thoroughly understand the behavior of ISOMAP in terms of parameter values.

The rest of this paper is organized as follows. Section 2 briefly introduces ISOMAP and its algorithm,

[1] The complexity of the (all-pairs) shortest path computation depends on the algorithm. Floyd-Warshall algorithm requires $O(n^3)$ computations while Dijkstra's algorithm does $O(|e|n \log n)$ computations [3] in which $|e|$ is the number of edges.

[2] It may not be possible to visualize the manifold curvature perfectly without using the original dimension, but at least we can obtain some insights about it from a lower dimensional visualization.

and Section 3 discusses previous work related to p-ISOMAP. Section 4 describes the algorithmic details and the complexity analysis of p-ISOMAP, and Section 5 presents not only the experimental results that compare the computation times of ISOMAP and p-ISOMAP but also interesting visualization examples of real-world data using p-ISOMAP. Finally, Section 6 concludes our work.

2 ISOMAP

Given a set of data points represented as M-dimensional vectors $x_i \in \mathbb{R}^M$ for $i = 1, \ldots, n$, ISOMAP assumes a lower dimensional manifold structure in which the data are embedded. It yields the m-dimensional representation of x_i as $y_i \in \mathbb{R}^m$ ($m \ll M$) such that the Euclidean distance between y_i and y_j approximates their geodesic distance along the underlying manifold as much as possible. Such an approximation builds on the classical MDS framework, but unlike MDS, ISOMAP has the capability of handling nonlinearity existing in the original space since a geodesic distance reflects an arbitrary curvilinear shape of the manifold.

On input, ISOMAP takes a data matrix $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \in \mathbb{R}^{M \times n}$, a reduced dimension m, and a parameter k or ϵ. The algorithm is composed of three steps:

1. *Neighborhood graph construction.* ISOMAP first computes the pairwise Euclidean distance matrix, $D_X \in \mathbb{R}^{n\times n}$, in which $D_X(i, j)$ is the Euclidean distance between x_i and x_j. Then it determines the set of neighbors for each point either by k-nearest neighbors or by those within a fixed radius ϵ. Between a point x_i and each of its neighbors x_j, an edge $e(i, j)$ is assigned with a weight equivalent to their Euclidean distance, and in this way, ISOMAP forms a weighted undirected neighborhood graph $G = (V, E)$, where the vertices in V correspond to the data points x_i's.

2. *Geodesic distance estimation.* In the second step, ISOMAP estimates the pairwise geodesic distance based on the shortest path length for every vertex pair along the neighborhood graph G, which is represented as a matrix $D_G \in \mathbb{R}^{n\times n}$ in which $D_G(i, j)$ is the shortest path length between x_i and x_j in G.

3. *Lower dimensional embedding.* The final step performs classical MDS on D_G, producing m-dimensional data embedding, $Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} \in \mathbb{R}^{m\times n}$. First, the pairwise geodesic distance matrix D_G is converted to an inner product matrix B_G as
(2.1)
$$B_G = -\left(I - \frac{1}{n}\mathbf{1}\mathbf{1}^\mathbf{T}\right) D_G.^2 \left(I - \frac{1}{n}\mathbf{1}\mathbf{1}^\mathbf{T}\right)/2,$$
in which $I \in \mathbb{R}^{n\times n}$ is an identity matrix, $\mathbf{1} \in \mathbb{R}^{n\times 1}$ is a vector whose elements are all 1's, and $D_G.^2$ is an element-wise squared D_G. Classical MDS solves Y such that it minimizes $E = \|B_G - Y^T Y\|$, where the matrix norm $\|\cdot\|$ is either a Frobenius or Euclidean norm. Such a solution of Y is obtained by the eigendecomposition of B_G as
$$(2.2)\quad Y = \begin{bmatrix} \sqrt{\lambda_1}v_1 & \sqrt{\lambda_2}v_2 & \cdots & \sqrt{\lambda_m}v_m \end{bmatrix}^T,$$
where $\lambda_1, \ldots, \lambda_m$ are the m largest eigenvalues of B_G, with corresponding eigenvectors $v_1, \ldots, v_m$.

Table 1: Notations used in this paper

Notation	Description
n	Number of data points
M	Original dimension
m	Reduced dimension
x_i	Input data vector, $i = 1, \ldots, n$
y_i	Reduced dimensional vector of x_i
D_X	Euclidean distance matrix of x_i's
$\overrightarrow{G}$	Directed neighborhood graph
G	(Undirected) neighborhood graph
q	Maximum degree of the vertices in G
D_G	Shortest path length matrix in G
B_G	Inner product matrix obtained from D_G
$\mathcal{A}$	Set of edges to be inserted in G
$\mathcal{D}$	Set of edges to be removed in G
Δe_i	Set of inserted/removed edges of x_i
F	Set of affected vertex pairs by $\mathcal{A}$ or $\mathcal{D}$
P	Predecessor matrix
H	Hop number matrix
k and k^{new}	Previous and new parameter values

3 Related Work

Based on the algorithmic details of ISOMAP described in Section 2, if the parameter k or ϵ varies, it would change the topology of the neighborhood graph in the first step. Such a change can be interpreted as either an insertion of new edges or a removal of existing edges in the neighborhood graph. The inserted or removed edges would in turn influence the shortest path length matrix D_G. Solving the updated D_G can be viewed as a dynamic shortest path problem in which we need to update the existing shortest path and its corresponding length due to graph changes. Generally, a dynamic shortest path problem includes all the various situations that involve not only vertex insertion/removal but also real-valued edge weight changes rather than just edge insertion/removal, and depending on what types of changes in the graph are assumed in the algorithm, it maintains a variety of additional information such as the candidates of the future shortest paths for an efficient shortest path update [4, 7, 9].

In the context of manifold learning methods, several approaches have tried to dynamically update ISOMAP embedding for incremental data input such as a data stream [11, 12]. Similar to the parameter changes in p-ISOMAP, incremental data cause topology changes in the neighborhood graph, which can be fully expressed by edge insertion/removal, so previous studies[11, 17] have discussed the dynamic shortest path updating algorithms that can deal with such types of graph changes. However, the characteristic in terms of graph changes differs greatly between p-ISOMAP and incremental ISOMAP. First, each parameter change in p-ISOMAP involves only the form of either edge insertion or removal in p-ISOMAP while a new data point causes both at once in incremental ISOMAP. In this sense, one may regard the shortest path updating in p-ISOMAP as simpler than that in incremental ISOMAP. However, graph changes in incremental ISOMAP primarily result from a new data item, and thus, an inserted edge in incremental ISOMAP is always connected to the new data point once an edge of a certain vertex is deleted. Furthermore, when the parameter k is used, the number of inserted or removed edges in p-ISOMAP is roughly $O(n|\Delta k|)$, where n is the number of data points and $\Delta k = k^{new} - k$, whereas that in incremental ISOMAP is roughly $O(k)$, which is much smaller than that in p-ISOMAP. This difference implies that even a small change in parameter values in p-IOSMAP can lead to a significant change in the neighborhood graph and its all-pairs shortest path results. However, the graph change in incremental ISOMAP is still minor compared to the entire graph size. Considering such different behaviors, we present our own shortest path updating algorithm that is appropriate for p-ISOMAP in Section 4.

After the shortest path update, one needs to update the eigendecomposition results on a new matrix B_G, shown in Eq. (2.1). In general, the eigendecomposition update is done by formulating the change in B_G in a certain form, e.g., a rank-1 update [5]. In incremental ISOMAP, [11] applied an approximation technique called the Rayleigh-Ritz method [10, 8] based on a variant of the Krylov subspace in computing the eigendecomposition. However, this method is limited to the case when the reduced dimension m is fairly large and the eigendecomposition does not change significantly. However, when p-ISOMAP is used in a visual analytics system, which is one of our main motivations, it requires

Algorithm 1 neighborhood graph update for a new k

Input: the new value of k , the directed neighborhood graph $\overrightarrow{G}$, and the undirected one G
Output: the set of inserted edges $\mathcal{A}$ or that of removed ones $\mathcal{D}$ in G, updated $\overrightarrow{G}$ and G

```
1: for all data point $x_i$ do
2:   for all newly added/deleted neighbor $x_j$ do
3:     Assign/Remove an edge $e(i, j)$ from $x_i$ to $x_j$ in $\overrightarrow{G}$.
4:   end for
5: end for
6: Initialize $\mathcal{A} := \emptyset$ / $\mathcal{D} := \emptyset$.
7: for all inserted/removed edge $e(i, j)$ in $\overrightarrow{G}$ do
8:   if $e(i, j)$ is an inserted edge then
9:     if $e(i, j)$ is not in $G$ then
10:      $\mathcal{A} \leftarrow \mathcal{A} \cup \{e(\min(i, j), \max(i, j))\}$
11:      Assign the edge $e(i, j)$ in $G$.
12:    end if
13:  else {$e(i, j)$ is a removed edge}
14:    if $e(j, i)$ is not in $\overrightarrow{G}$ then
15:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{e(\min(i, j), \max(i, j))\}$
16:      Remove the edge $e(i, j)$ in $G$.
17:    end if
18:  end if
19: end for
```

m to be a small value such as two or three. Furthermore, such approximation methods perform poorly in p-ISOMAP since it involves $O(n)$ graph changes and the corresponding large amount of the shortest path update. Hence, we focus on the exact solution for p-ISOMAP.

In the next section, we present a novel algorithmic framework for p-ISOMAP.

4 p-ISOMAP

p-ISOMAP assumes the original ISOMAP result is available for a particular parameter value. Given a new parameter value, the algorithm performs three steps: the neighborhood graph update, the shortest path update, and the eigenvalue/vector update.

4.1 Neighborhood Graph Update In this step, p-ISOMAP computes the set of edges to insert or remove from the previous neighborhood graph and update the neighborhood graph by applying such changes. In order to compute these edges efficiently, each data point maintains the sorted order of the other points in terms of its Euclidean distances to them. In this way, p-ISOMAP identifies which neighbor points of a particular point are to be added or deleted in $O(1)$ time.

If a neighborhood graph is constructed by the parameter ϵ, the neighborhood relationship is symmetric, i.e., if and only if x_i is a neighbor of x_j, x_j is also a neighbor of x_i for a particular ϵ. Thus the added or deleted neighborhood pairs are equivalent to the inserted or removed edges in a neighborhood graph, and the algorithm is straightforward.

On the other hand, if a neighborhood graph is constructed by k-NN, the situation becomes complex since it is possible that x_i is a neighbor of x_j, but x_j is not a neighbor of x_i, which we call a one-sided neighborhood. By considering such a one-sided relationship, a directed neighborhood graph $\overrightarrow{G}$ is initially made, and ISOMAP obtains its undirected one G by an OR operation, i.e., for x_i and x_j, if at least either one is a neighbor of the other, then ISOMAP assigns an edge with the weight equal to their Euclidean distance. In p-ISOMAP, both directed and undirected graphs are maintained and updated in an orderly manner so as to avoid ambiguity about which changes of neighbors in a directed graph cause actual edge changes in an undirected one in which we have to actually compute the shortest paths. The detailed procedure of the neighborhood graph update are described in Algorithm 1. As an output, it produces the set of effectively inserted/removed edges, which is, in turn, used in the shortest path update stage.

4.1.1 Time Complexity In ISOMAP, the time complexity in constructing a neighborhood graph is as follows. It starts with a sort operation for a given data set whose time complexity is $O(n^2 \log n)$. Then obtaining $\overrightarrow{G}$ and G requires $O(nq)$, in which q is the maximum degree of vertices in the graph G. In p-ISOMAP, the time complexity required in the neighborhood graph update is bounded by $O(n \cdot \max_i |\Delta e_i|)$, in which $|\Delta e_i|$ is the number of inserted/removed edges associated with x_i.

4.2 Shortest Path Update The shortest path update stage, which is one of the most computationally intensive steps in p-ISOMAP, takes the input as either $\mathcal{A}$ or $\mathcal{D}$ and updates the shortest path length matrix D_G. In order to facilitate this process, p-ISOMAP maintains and updates the information about the shortest path itself with a minimal memory requirement in the form of a predecessor matrix $P \in \mathbb{R}^{n\times n}$, in which $P(i, j)$ stores the node index immediately preceding x_j in the shortest path from x_i to x_j.[3] For instance, if the shortest path from x_1 to x_2 is composed of $x_1 \rightarrow x_4 \rightarrow x_3 \rightarrow x_2$, then we set $P(1, 2) = 3$.

For the shortest path update, p-ISOMAP performs two steps:

1. It identifies the set, F, of the "affected" vertex pairs, whose shortest paths need to be recomputed due to the inserted edges in $\mathcal{A}$ or the removed edges in $\mathcal{D}$.

2. Then it computes their shortest paths based on the information of the rest of the vertex pairs and the newly updated neighborhood graph, which usually performs significantly faster than the original shortest path computation from scratch.

Algorithm 2 Shortest path update for $\mathcal{A}$

```
Input: the updated neighborhood graph $G$, the shortest path length matrix $D_G$, the predecessor matrix $P$, and the set of inserted edges $\mathcal{A}$
Output: updated $D_G$ and $P$
 1: for all inserted edge $e(a, b)$ in $\mathcal{A}$ do
 2:   for all data point $x_i$ do
 3:     Unmark all the other nodes except for $x_i$.
 4:     if $D_G(i, b) + G(a, b) < D_G(i, a)$ then
 5:       $D_G(i, a) \leftarrow D_G(i, b) + G(a, b)$
 6:       $P(i, a) \leftarrow b$
 7:       $D_G(a, i) \leftarrow D_G(i, a)$
 8:       if $b = i$ then
 9:         $P(a, i) \leftarrow a$
10:       else
11:         $P(a, i) \leftarrow P(b, i)$
12:       end if
13:     end if
        {Traverse $T(i, a)$}
14:     Initialize an empty queue $Q$.
15:     $Q$.enqueue($a$)
16:     while $Q$ is not empty do
17:       $t := Q$.pop
18:       Mark $x_t$.
19:       for all unmarked node $x_j$ adjacent to $x_t$ do
20:         if $D_G(i, t) + G(t, j) < D_G(i, j)$ then
21:           $D_G(i, j) \leftarrow D_G(i, t) + G(t, j)$
22:           $P(i, j) \leftarrow t$
23:           $D_G(j, i) \leftarrow D_G(i, j)$
24:           $P(j, i) \leftarrow P(t, i)$
25:           $Q$.enqueue($j$)
26:         else
27:           Mark $x_j$.
28:         end if
29:       end for
30:     end while
31:   end for
32: end for
```

[3] Here we assume the shortest path is unique for every vertex pair, which is almost always the case in ISOMAP.

4.2.1 Shortest Path Update with Inserted Edges due to an Increasing Parameter

The main idea in the shortest path update due to an inserted edge $e(a, b)$ is that if $D_G(i, a) + e(a, b) + D_G(b, j)$ or $D_G(i, b) + e(a, b) + D_G(a, j)$ is shorter than $D_G(i, j)$, then $D_G(i, j)$ is to be replaced by the smaller one between the two new path lengths along with the corresponding update of P. Performing this comparison for all pairs of vertices would require the time complexity of $O(n^2|\mathcal{A}|)$, in which $|\mathcal{A}|$ is the number of edges in $\mathcal{A}$. Unlike incremental ISOMAP or other situations in which $|\mathcal{A}|$ is relatively small and constant, p-ISOMAP has $|\mathcal{A}| \simeq O(n)$ due to an increasing parameter, which makes the complexity of the above computation roughly equal to $O(n^3)$. Such complexity is no better than the Floyd-Warshall algorithm used in the original ISOMAP. Thus, the algorithm has to find the computational gain while identifying the subset, F, of the entire vertex pair set and applying the above comparison only in this set. For construction of F, the shortest path can conveniently be interpreted as a form of a tree in which $T(i)$ is the shortest path tree that has x_i as its root. The subtree $T(i; a)$ of $T(i)$ can then be defined as one with a root at x_a. Using a well-known property that any subpaths of the shortest path are also the shortest path, once a new shortest path from a particular vertex x_i to x_a is found using $e(a, b)$, one can traverse $T(i; a)$ in various ways such as a breath-first-search or a depth-first-search method and correspondingly update the shortest paths from x_i to the nodes in $T(i; a)$. In p-ISOMAP, we have used the breath-first-search, the detailed algorithm of which is summarized in Algorithm 2.

In fact, such approaches using subtree traversal for inserted edges were applied in many applications [14, 17]. However, the algorithm presented here was found simpler and faster since it deals with both directional paths at once when updating D_G and P.

4.2.2 Shortest Path Update with Deleted Edges due to a Decreasing Parameter

When edges are deleted, the vertex pairs in F are those whose shortest paths include any of these deleted edges. The set F can be identified by considering deleted edges one by one and then by performing union operation on such vertex pair sets. This approach is reasonable when $|\mathcal{D}|$ is small and thus little overlap occurs between such vertex pair sets for each deleted edge, which is the case in incremental settings [11, 17]. In contrast, p-ISOMAP has $|\mathcal{D}| = O(n)$, which possibly leads to a large amount of overlap in the affected vertex pairs among different deleted edges; therefore, the above approach results in a significantly redundant computation.

For this reason, we propose a new algorithm for p-ISOMAP that identifies the affected vertex pairs by handling all the deleted edges at once. The key idea is that for the shortest path between a particular vertex pair, we partition it into multiple subpaths separated by any deleted edges, and then we form Cartesian products between any two such subpaths and place them in F. For example, when the shortest path is $x_1 \to x_3 \to x_2 \to x_4$, if $e(x_3, x_2)$ is deleted, we add $\{(x_i, x_j)|i \in \{1, 3\}, j \in \{2, 4\}\}$ to F.

Furthermore, we enhance the efficiency in this process in the following way. We first consider the vertex pair whose shortest path has the largest number of hops and check all of its subpaths, which are also the shortest paths between their hopping nodes. Then, the checked vertex pairs are not considered again. In other words, by first dealing with the shortest paths that cover as many other shortest paths as possible, we can handle the maximum number of vertex pairs regarding whether they are to be added to F or not. To implement this idea, we maintain a hop number matrix H in which $H(i, j)$

Algorithm 3 Identification of F due to $\mathcal{D}$

Input: the removed edge set $\mathcal{D}$, the predecessor matrix P, and the hop number matrix H
Output: the set of the affected vertex pairs F

```
1: $\alpha := \max_{i,j} H_{ij}$
2: Initialize a linked list $l[h]$ for $h = 1, \ldots, \alpha$
3: Unmark all vertex pairs $(x_i, x_j)$ such that $i < j$
4: for all vertex pair $(x_i, x_j)$ such that $i < j$ do
5:     Insert $(x_i, x_j)$ to $l[H(i, j)]$.
6: end for
7: for $h := \alpha$ to 1 do
8:     for all Unmarked vertex pair $(x_i, x_j)$ in $l[h]$ do
9:         Set $p[k]$ for $k = 1, \ldots, h$ as $k$-th node found
           in the shortest path from $x_j$ to $x_i$
10:        for $k := 1$ to $h$ do
11:            $m[k] := \max_{k \leq l \leq h}(l)$ such that a vertex pair
               $(p[k], p[l])$ is marked.
12:            Mark vertex pairs $(p[k], p[v])$ and $(p[v], p[k])$
               for all $v$ such that $m[k] + 1 \leq v \leq h$
13:        end for
14:        $q := 1$
15:        for $k := h - 1$ to 1 do
16:            if $(p[k], p[k+1]) \in \mathcal{D}$ then
17:                Insert vertex pairs $(p[u], p[v])$, for all $u$
                   and $v$ such that $q \leq u \leq k$ and $\max(k +
                   1, m[u] + 1) \leq k \leq h$, to $F$.
18:                $q \leftarrow k + 1$
19:            end if
20:        end for
21:    end for
22: end for
```

Algorithm 4 Recomputation of the shortest paths for F for a decreasing parameter

```
Input: the updated graph $G$, the shortest path length matrix $D_G$, the predecessor matrix $P$, the hop number matrix $H$, and the set of the affected vertex pairs $F$
Output: updated $D_G$, $P$, and $H$
 1: Sort vertices in terms of how frequently they appear in $F$ in an increasing order
 2: for all $x_i$ in the above sorted order do
 3:   Initialize an empty heap $Q$
 4:   $C := \{x_j | (x_i, x_j) \in F\}$
 5:   for all $x_j \in C$ do
 6:     $d := \infty$
 7:     for all adjacent node $x_t$ of $x_j$ such that $x_t \notin C$ do
 8:       if $d > D_G(i, t) + G(t, j)$ then
 9:         $d \leftarrow D_G(i, t) + G(t, j)$
10:         $P(i, j) \leftarrow t$
11:         $H(i, j) \leftarrow H(i, t) + 1$; $H(j, i) \leftarrow H(i, j)$
12:       end if
13:     end for
14:     Insert an entry $(d, j)$ with a key $d$ and an index $j$ to $Q$
15:   end for
16:   while $Q$ is not empty do
17:     $(d, j)$ :=ExtractMin from $Q$
18:     $D_G(i, j) \leftarrow d$; $D_G(j, i) \leftarrow d$
19:     Remove $(x_i, x_j)$ from $C$ and $F$
20:     $pred := j$
21:     while $P(i, pred) \neq i$ do
22:       $pred \leftarrow P(i, pred)$
23:     end while
24:     $P(j, i) \leftarrow pred$
25:     for all adjacent node $x_t$ of $x_j$ such that $x_t \in C$ do
26:       $d :=$ a key of an entry with an index $t$ in $Q$
27:       if $d > D_G(i, j) + G(j, t)$ then
28:         DecreaseKey $(D_G(i, j) + G(j, t), t)$ in $Q$
29:         $H(i, t) \leftarrow H(i, j) + 1$; $H(j, i) \leftarrow H(i, t)$
30:       end if
31:     end for
32:   end while
33: end for
```

contains the number of hops in the shortest path from vertex i to j, which enables us to prioritize the vertex pair according to its number of hops. In addition, our algorithm takes into account overlapping subpaths between the shortest paths of different vertex pairs. That is, if any subpaths of the shortest path of a certain vertex pair are also those of another vertex pair that has been previously taken care of, the algorithm stops checking such subpaths. In this way, we completely exclude redundant computations in an efficient manner. The detailed algorithm to solve for the set F is described in Algorithm 3.

Once F is obtained, the shortest paths are recomputed selectively. This process can be expedited using the available information about the unaffected vertex pairs whose shortest paths remain unchanged. We choose Dijkstra's algorithm as the main algorithm for the shortest path computation since it is suitable for a sparse graph. How to incorporate the above available information in Dijkstra's algorithm is straightforward as described in Algorithm 4. In addition, since Dijkstra's algorithm is a single-source shortest path algorithm, it needs to run n times for each source vertex. In terms of the order of the source vertices on which to run Dijkstra's algorithm, those that have the least number of destination nodes to update are processed first, and the updated vertex pairs are then removed from F. Algorithm 4, which also includes additional functionalities for updating P and H, summarizes the shortest path update process based on F.

4.2.3 Time Complexity When a parameter increases, Algorithm 2 requires the time complexity of $O(|\mathcal{A}|nq \cdot \max_{i,a} |T(i;a)|)$ in which $\max_{i,a} |T(i;a)|$ is the maximum number of nodes in subtree $T(i;a)$ over all x_i's and inserted edge $e(a, b)$'s. This complexity can be loosely bounded by $O(|\mathcal{A}|q|F|)$ where $|F|$ is the number of affected vertex pairs due to the inserted edges in $\mathcal{A}$.

For a decreasing parameter, the time complexity of Algorithm 3 requires $O(n^2)$ computations since it visits every vertex pair exactly once. Now, let us partition the entire vertices into two disjoint sets $V_d(i)$ and $V_d^c(i)$ such that $V_d(i) = \{x_j | (x_i, x_j) \in F\}$ for a certain x_i. Then, the complexity of Algorithm 4 is represented as $O(n \cdot \max_i(|E_i^{'}| \log |V_d(i)| + (|E_i^{''}|))$ in which $E_i^{'} = \{e(x_a, x_b) \in G | x_a, x_b \in V_d(i)\}$ and $E_i^{''} = \{e(x_a, x_b) \in G | x_a \in V_d(i), x_b \notin V_d(i)\}$.

In both cases, for small changes in the neighborhood graph, $|F|$ is expected to be much smaller than n^2, which is the maximum possible value of $|F|$.

4.3 Eigenvalue/vector Update Let us denote the updated D_G after the shortest path update described in Section 4.2 as D_G^{new}. In this step, D_G^{new} is first converted into the pairwise inner product matrix B_G^{new} by Eq. (2.1). To get a lower dimensional embedding as shown in Eq. (2.2), we need to obtain m eigenvalue/vector pairs $(\lambda_1^{new}, v_1^{new})$, $\ldots$, $(\lambda_m^{new}, v_m^{new})$ for B_G^{new}. In this computation, the available information that we can exploit is the previous m eigenvalue/vector pairs $(\lambda_1, v_1), \ldots, (\lambda_m, v_m)$ of B_G. In fact, they can be good

Table 2: Computation time in seconds between ISOMAP and p-ISOMAP. In parentheses next to the data set name, the three numbers are the number of data n, the original dimension M, and the reduced dimension m, respectively. The number in the other parentheses next to k value changes indicates the ratio of vertex pairs whose shortest paths need to be updated. For each case, the average computing times of 10 trials were presented.

Synthetic data	Rand (3500, 5000 $\to$ 50)				Swiss roll (4000, 3 $\to$ 2)			
	ISOMAP		p-ISOMAP		ISOMAP		p-ISOMAP	
$k \to k^{new}$ ($\lvert F\rvert/n^2$)	28	32	30 $\to$ 28 (15%)	30 $\to$ 32 (13%)	14	16	15 $\to$ 14 (77%)	15 $\to$ 16 (73%)
Neighborhood graph	1.6	1.6	0.1	0.1	1.8	1.8	0.2	0.2
Shortest path	12.3	12.6	5.1	4.7	16.4	17.1	17.3	15.6
Eigendecomposition	7.8	7.7	6.9	6.8	1.9	1.7	1.6	1.5
Real-world data	Pendigits (3000, 16 $\to$ 5)				Medline (2500, 22095 $\to$ 200)			
	ISOMAP		p-ISOMAP		ISOMAP		p-ISOMAP	
k	46	54	50 $\to$ 46 (39%)	50 $\to$ 54 (36%)	37	43	40 $\to$ 37 (21%)	40 $\to$ 43 (19%)
Neighborhood graph	1.3	1.2	0.1	0.1	1.1	1.1	0.1	0.1
Shortest path	9.3	9.8	7.1	8.3	6.8	7.0	2.8	3.3
Eigendecomposition	2.3	2.3	2.0	2.1	16.3	16.4	15.1	15.1

Table 3: Computation time in seconds to take to determine the optimal k value by minimizing residual variances.

	Rand	Swiss roll	Pendigits	Medline
Range of k	[5, 50]	[5, 50]	[7, 60]	[9, 70]
ISOMAP	580	635	692	776
p-ISOMAP	142	403	305	314

initial guesses for m eigenvalue/vector pairs for B_G^{new}, assuming the two matrices B_G and B_G^{new} are not much different in any sense.

The original ISOMAP uses the Lanczos algorithm [10], which is an iterative method that is appropriate for solving the first few leading eigenvalue/vector pairs. The Lanczos algorithm iteratively refines the solution in the Krylov subspace that grows from an initial vector by multiplying it with the matrix, i.e., span(b, $B_G^{new}b$, $(B_G^{new})^2b$, $\ldots$). The performance of the Lanczos algorithm largely depends on how quickly such a Krylov subspace covers that spanned by the eigenvectors. Another characteristic of the Lanczos algorithm is that the least leading eigenvalue/vector pair converges slowest within a particular tolerance. In other words, when the Krylov subspace becomes k dimensions, the first leading eigenvalue is refined k times, the second one $(k-1)$ times, the third one $(k-2)$ times, and so on.

In this sense, we suggest using an initial vector from which the Krylov subspace grows as v_m, i.e., span(v_m, $B_G^{new}v_m$, $(B_G^{new})^2v_m$, $\ldots$), which possibly best recovers (λ_m^{new}, v_m^{new}). As a result, we can expect the Lanczos algorithm to terminate in less number of iterations than in any other cases.

5 Experiments and Applications

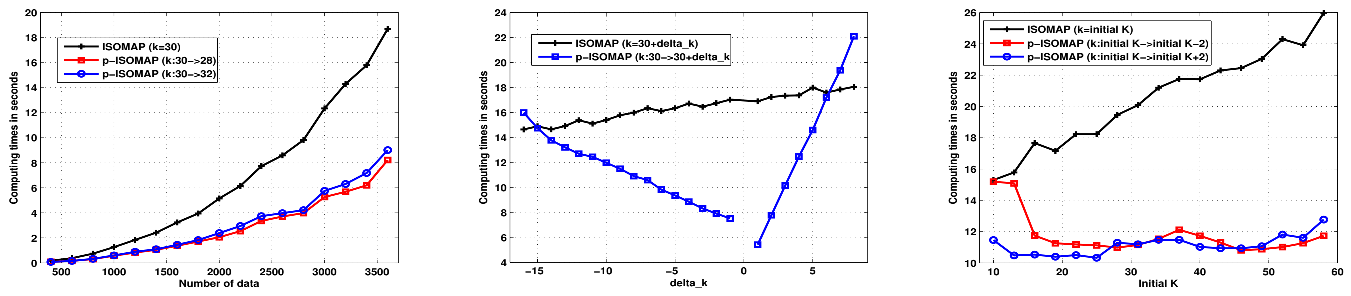
In this section, we present an empirical comparison between the computation times of ISOMAP and those of p-ISOMAP using both synthetic and real-world data sets. In addition, we show visualization applications of p-ISOMAP for real-world data sets.

In our experiments, we used the code of ISOMAP provided by the original author.[4] However, the original code does not take advantage of sparse graphs, so we compared p-ISOMAP with an improved version of ISOMAP that runs Dijkstra's algorithm in C++ with a sparse representation of the graph. p-ISOMAP was implemented mainly in MATLAB except for the shortest path update part, which runs in C++. In both ISOMAP and p-ISOMAP, the eigendecomposition was done by MATLAB built-in function "eigs," which performs the Lanczos algorithm by using Fortran library ARPACK [13].

Throughout all experiments, we used the ISOMAP parameter as k, where the neighborhood graph is constructed by k-NN, since we can easily bound $|\mathcal{A}|$ or $|\mathcal{D}|$ by $O(n\Delta k)$ in which $\Delta k = k^{new} - k$. All the experiments were done using MATLAB 7.7.0 on Windows Vista 64bit with 3.0GHz CPU with a 4.0GB memory.

5.1 Computation Time To compare the computation times between ISOMAP and p-ISOMAP, we tested two synthetic data sets (Rand and Swiss roll) and two real-world data sets (Pendigits and Medline). Rand data set was made by sampling a uniform distribution in a 5,000-dimensional hypercube, $[0, 1]^{5000}$, where the number of data is 3,500. "Swiss roll" data set has 4,000 data points in three-dimensional space. Pendigits data

[4] `http://waldron.stanford.edu/~isomap/IsomapR1.tar`



(a) Computing times vs. number of data (b) Computing times vs. Δk (c) Computing times vs. initial k

Figure 2: Behavior of p-ISOMAP depending on the number of data, Δk , and initial k on Rand data set. Other than the varied one, the rest of variables were fixed in each figure.

set[5] contains 10,992 handwritten digit data in a form of pen traces in 16-dimensional space [1], but we selected 3,000 data with an equal number of data per cluster because of memory constraints. Finally, Medline data set[6] is a document corpus related to medical science from the National Institutes of Health, and it has 2,500 documents encoded in 22,095-dimensional space.

Table 2 compares computation times of ISOMAP with those of p-ISOMAP for each data set. In most cases, p-ISOMAP runs significantly faster than ISOMAP. However, as the number of vertex pairs whose shortest paths need to be updated increases, the computational advantage of p-ISOMAP over ISOMAP gradually vanishes. Nonetheless, except for "Swiss roll" data set, which involves a large number of the shortest path update even with a slight parameter change, most data sets require only about 10-40% the shortest path update for a reasonable parameter change, e.g., within 5.

Figure 2 shows the behaviors of p-ISOMAP depending on the number of data, Δk, and an initial k value. We selected Rand data since it was the most suitable one to clearly observe its behaviors. Figure 2(a) shows the computation time in terms of the number of data. As we can see, p-ISOMAP scales well in terms of the number of data compared to ISOMAP. In Figure 2(b), as the parameter change Δk gets bigger, the running time of p-ISOMAP increases linearly, which tells that $|\mathcal{A}|$ or $|\mathcal{D}|$, which is proportional to Δk, has a dominant influence on the performance of p-ISOMAP. Finally, Figure 2(c) shows an increasing performance gap between two methods as an initial k value grows. This is mainly because the original Dijkstra's algorithm used in ISOMAP needs more computations as the graph gets denser while p-ISOMAP depends only on $|\mathcal{A}|$, $|\mathcal{D}|$, or correspondingly $|F|$, which probably does not increase over different initial k values.

Finally, for each data set, we measured the computation times to take to determine the optimal k value that minimizes residual variances [2]. As shown in Table 3, we could significantly reduce the computation times by utilizing the dynamic update of p-ISOMAP.

5.2 Knowledge Discovery via Visualization using p-ISOMAP

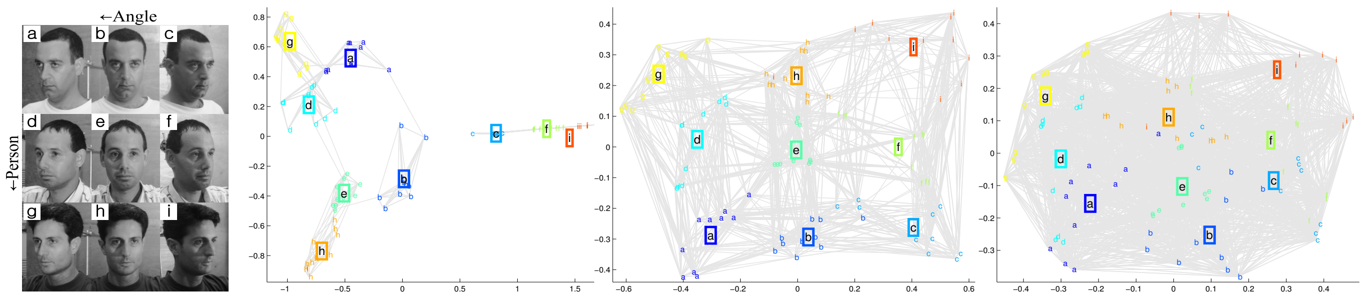
In this section, we present interesting visualization examples of real-world data sets using p-ISOMAP. To be specific, we show how ISOMAP with different parameters can discover various knowledge about data and how the information acquired through visualization can facilitate traditional data mining problems such as a classification task. p-ISOMAP was used to efficiently update ISOMAP results throughout all the visualization experiments.

To begin with, we have chosen three real-world data sets (Weizmann, Medline, and Pendigits) that have cluster structures in order to make it easy to analyze their visualization. Weizmann data set is a facial image data set[7] that has 28 persons' images with various angles, illuminations, and facial expressions. To obtain an understandable visualization, we have chosen three particular persons' images with three different viewing angles as shown in Figure 3(a), in which each combination of a particular person and a viewing angle contains multiple images that vary based on other factors such as illuminations and facial expressions. In their visualizations shown in Figures 3(b)-(d), each of these images is represented as a letter that corresponds to its cluster from Figure 3(a). Medline data set, which is a document collection, has 5 topic clusters, heart attack ('h'), colon cancer ('c'), diabetes ('d'), oral cancer ('o'), and tooth decay ('t'), in which the letters in parentheses are used in its visualization in Figure 4. Pendigits data set, which is described in Section 5.1, has 10 clusters in terms of which digit each

[5]http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

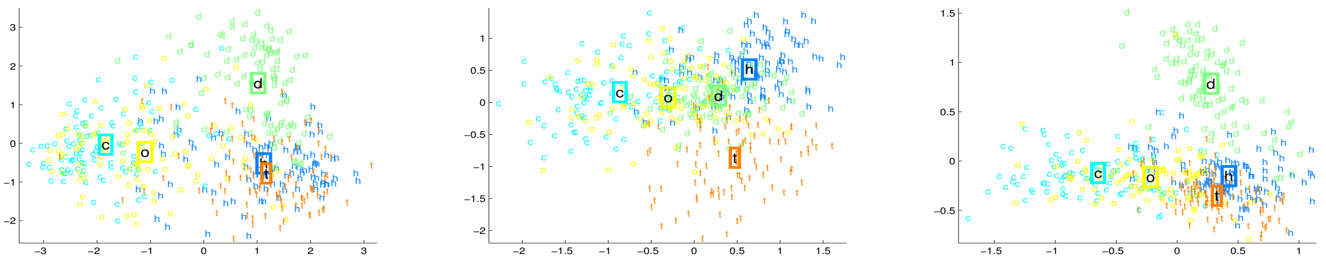
[6]http://www.cc.gatech.edu/~hpark/data.html

[7]ftp://ftp.wisdom.weizmann.ac.il/pub/facebase



(a) Cluster structure (b) $k = 8$ (c) $k = 22$ (d) $k = 49$

Figure 3: Visualization of Weizmann data set using p-ISOMAP



(a) $k = 5$ (b) $k = 30$ (c) $k = 50$

Figure 4: Visualization of Medline data set using p-ISOMAP

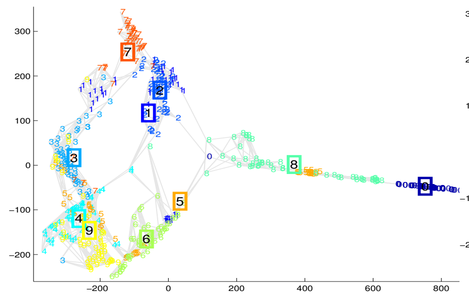
data item corresponds to, i.e., '0', '1', ..., '9'. Several interesting visualization examples of these data based on p-ISOMAP are shown in Figures 3-5[8] where cluster centroids and neighborhood connections are also shown in the form of letters in rectangles and grey lines in the background, respectively.

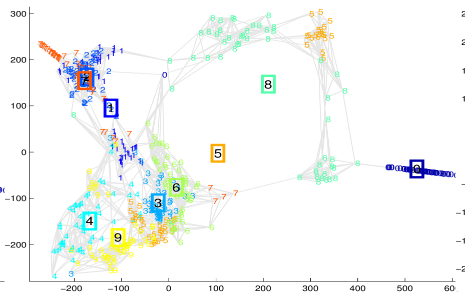
Among visualization examples of Weizmann data set, Figure 3(c), which well resembles the layout of clusters in Figure 3(a), successfully straightens its intrinsic manifold defined by the two factors, a person and an angle. This is mainly because of the neighborhood graph constructed by a proper k value that forms its edges either within a particular person or within a particular angle, which is why we mostly see horizonal and vertical neighborhood connections as well as gaps between grid-shaped cluster centroids in Figure 3(c). Regarding a comparison between Figures 3(b) and 3(d), fewer neighbors in Figure 3(b) bring connections only within images with the same angle, which in turn results in a clustered form of visualization based on angles. This indicates that even if we prefer the similarity in terms of a person to that in terms of an angle, the actual distances in the vector space into which the images are transformed are dominated by an angle. On the other hand, Figure 3(d) connects almost all the data points between each other, which would reflect the Euclidean distances in the original space just like MDS does. In addition, we can consider the layout of cluster structure shown in Figure 3(d) as a curved version of manifold as it appears in the original space, which is analogous to what we discussed in Figure 1.

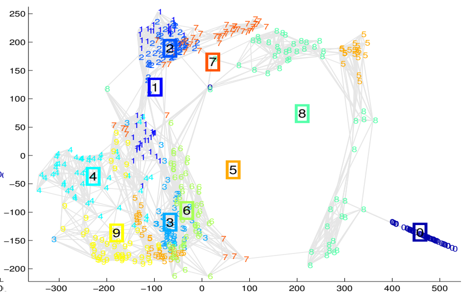
Medline data shown in Figure 4 is not visualized in a well-clustered form by ISOMAP because it is usually difficult to find a well-defined manifold structure with few meaningful dimensions for document data. However, by manipulating k values, we can at least obtain various visualization results that possibly reveal different aspects of the data. For example, when $k = 30$ in Figure 3(b), the topic cluster, tooth decay ('t'), is shown distinct from the other clusters while so does the cluster, diabetes ('d'), in the other cases. In this situation, if one wants to focus on a certain cluster separately from the others, it would be necessary to change k values for a suitable visualization result.

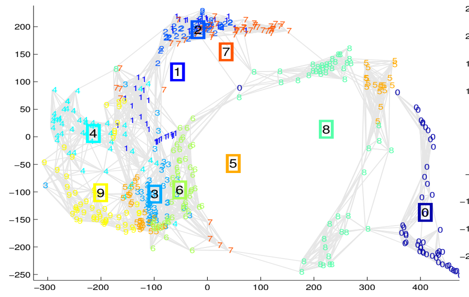
Visualizations of Pendigits data set shown in Figure 5 give numerous interesting characteristics. First of all, as the parameter k increases, the overall transition from Figure 5(a) to 5(f) is shown similar to that of "Swiss roll" data set from Figure 1(c) to 1(d). In other words, a larger k value places more data in a curved shape, which reflects the underlying curvature in the original space, while a smaller k value does more data in a linear shape, which corresponds to a straightened manifold. To be specific, starting from Figure 5(b), the cluster '8' gradually gets scattered and curved with an increasing

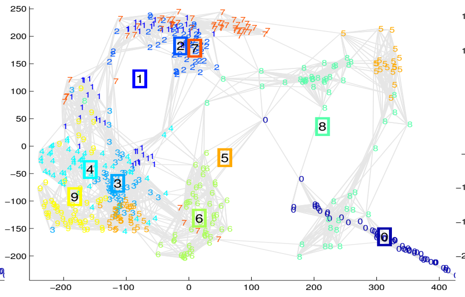
[8]These figures can be arbitrarily magnified without losing the resolution in the electronic version of this paper


(a) $k = 7$


(b) $k = 9$


(c) $k = 10$


(d) $k = 11$


(e) $k = 13$

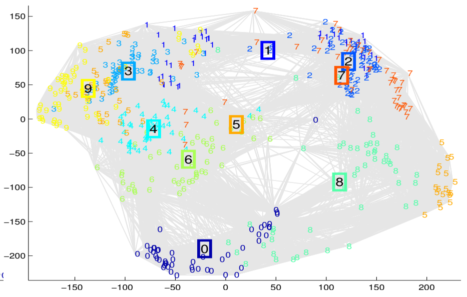
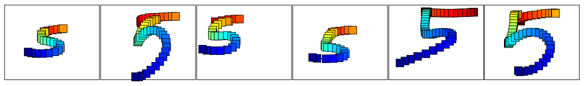
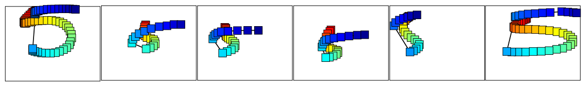
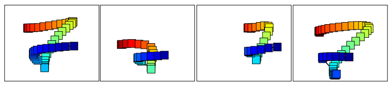
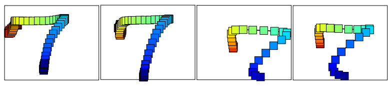

(f) $k = 50$

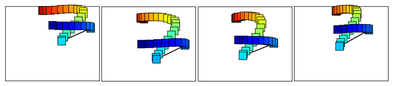
Figure 5: Visualization of Pendigits data set using p-ISOMAP

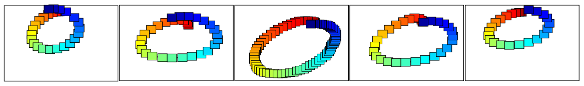

(a) A subcluster in cluster ‘5’


(b) Another subcluster in cluster ‘5’


(c) Major data in cluster ‘7’


(d) A minor group in cluster ‘7’


(e) Another minor group in cluster ‘7’


(f) Major data in cluster ‘0’

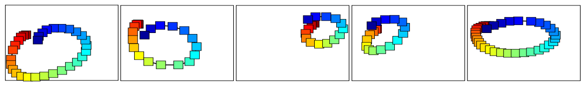

(g) Minor data in cluster ‘0’

Figure 6: Subclusters/outliers in ‘0’, ‘5’, and ‘7’. Pen traces start from red and end at blue.

k. Similarly, the cluster ‘0’ maintains a linear shape before $k = 50$, and finally it becomes scattered in Figure 5(f). In short, ISOMAP with a small parameter value tends to unroll the curved manifold due to geodesic paths, but that with a large parameter better shows its curvature itself. In view of clustering, Figure 5(a) well separates the clusters ‘2’ and ‘7’ whereas the other visualizations gradually overlap them with increasing k values. In addition, the clusters ‘3’ and ‘6’ appears to overlap for a certain range of k between 9 and 11 as shown in Figures 5(b)-(d).

Now let us discuss about subcluster/outlier discovery through various visualization examples. In most examples in Figure 5, the cluster ‘5’ is shown to have two subclusters, one of which is near the cluster ‘8’, and the other between the clusters ‘3’ and ‘9’. Based on this observation, we examined some sample data from each cluster and found out such subclusters are due to the different way to write ‘5’.[9] From the examples in these two subclusters shown in Figures 6(a)-(b), we can see that some people write ‘5’ starting from the hat, which is the top horizontal line in ‘5’, while others write the hat after finishing the bottom part. Similarly, the cluster ‘7’ has a majority of data near the cluster ‘2’, but it also has two minor groups of data near the cluster ‘1’ and the cluster ‘6’, respectively. (See, for example, the coordinates around $(-100, 50)$ and $(50, -150)$ in Figure 5(c).) After looking at the actual data samples from these groups, we found that most people write ‘7’ in a way shown in Figure 6(c). However, some people first write an additional small vertical line in the top-left part but by omitting the small horizontal line in the middle

[9]Note that Pendigits data set we used here is not just static image data but the traces of the pen, which is why the order matters in the feature space.

part as shown in Figure 6(d), which corresponds to the minor data near the cluster '1', but some others just reverse the direction to write the small horizontal line in the middle part of '7' as shown in Figure 6(e), which corresponds to those near the cluster '6'. In addition, their different traces and shapes impose similarities to those of the clusters '1' and '6', respectively. Finally, in Figure 5(d), some data in the cluster '0' seems to deviate from its major line-shaped data in Figures 5(a)-(c). Figures 6(f) and 6(g) represent the latter and the former data, respectively. We can see that such deviated ones shown in Figure 6(g) start from the top-right corner rather than from the top-middle part when writing '0', which causes their connections to the cluster '5' that also starts from the top-right corner.

Finally, we have incorporated the above findings in a handwritten digit recognition, which is a classification problem, using Pendigits data set. Based on the information that the cluster '5' has two clear subclusters, we modified the training data labels in the cluster '5' into two different labels and classified the test data that are assigned either label to the cluster '5'. As a classification method, we have chosen the linear discriminant analysis combined with k-nearest neighbor classification, which is a common setting in classification. As a result, the classification accuracy increased from 89% to 93%. In fact, this is a promising example of human-aided data mining processes through visualizations with intelligent interaction. The computational efficiency of p-ISOMAP makes such processes smooth and prompt.

6 Conclusions

In this paper, we proposed p-ISOMAP, an efficient algorithmic framework to dynamically update ISOMAP embedding for varying parameter values. The experiments using both synthetic and real-world data with various settings validate its efficiency. This advantage of p-ISOMAP can not only speed up the parameter optimization processes but also enable users to interact with visual analytics systems more smoothly. Such interaction provides us with deep understanding about data, which can improve even the computational data mining problems such as classification.

References

[1] A. Asuncion and D.J. Newman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2007.

[2] Mukund Balasubramanian, Eric L. Schwartz, Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. The Isomap Algorithm and Topological Stability. *Science*, 295(5552):7a, 2002.

[3] M. Barbehenn. A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. *Computers, IEEE Transactions on*, 47(2):263, Feb 1998.

[4] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record: Journal of the Transportation Research Board*, 1645(-1):170–175, 1998.

[5] B. Champagne. Adaptive eigendecomposition of data covariance matrices based on first-order perturbations. *Signal Processing, IEEE Transactions on*, 42(10):2758–2770, Oct 1994.

[6] Jaegul Choo, Shawn Bohn, and Haesun Park. Two-stage framework for visualization of clustered high dimensional data. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on*, pages 67–74, Oct. 2009.

[7] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

[8] J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[9] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251 – 281, 2000.

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

[11] M.H.C. Law and A.K. Jain. Incremental nonlinear dimensionality reduction by manifold learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(3):377–391, March 2006.

[12] M.H.C. Law, N. Zhang, and A.K. Jain. Nonlinear Manifold Learning For Data Stream. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 33–44. Society for Industrial Mathematics, 2004.

[13] R.B. Lehoucq, D.C. Sorensen, and C. Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. Society for Industrial and Applied Mathematics, 1998.

[14] Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and sql. *ACM Trans. Database Syst.*, 30(3):698–721, 2005.

[15] O. Samko, A.D. Marshall, and P.L. Rosin. Selection of the optimal parameter value for the isomap algorithm. *Pattern Recognition Letters*, 27(9):968 – 979, 2006.

[16] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[17] Dongfang Zhao and Li Yang. Incremental isometric embedding of high-dimensional data using connected neighborhood graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1):86–98, Jan. 2009.