Available online at www.sciencedirect.com
SCIENCE DIRECT®
Neurocomputing 67 (2005) 29–53
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Nonlinear dimensionality reduction of data manifolds with essential loops

John Aldo Lee*, Michel Verleysen[1]

Université Catholique de Louvain, Microelectronics Laboratory, Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium

Received 29 February 2004; received in revised form 1 September 2004; accepted 18 November 2004

Communicated by S. Fiori

Abstract

Numerous methods or algorithms have been designed to solve the problem of nonlinear dimensionality reduction (NLDR). However, very few among them are able to embed efficiently 'circular' manifolds like cylinders or tori, which have one or more essential loops. This paper presents a simple and fast procedure that can tear or cut those manifolds, i.e. break their essential loops, in order to make their embedding in a low-dimensional space easier. The key idea is the following: starting from the available data points, the tearing procedure represents the underlying manifold by a graph and then builds a maximum subgraph with no loops anymore. Because it works with a graph, the procedure can preprocess data for all NLDR techniques that uses the same representation. Recent techniques using geodesic distances (Isomap, geodesic Sammon's mapping, geodesic CCA, etc.) or K-ary neighborhoods (LLE, hLLE, Laplacian eigenmaps) fall in that category. After describing the tearing procedure in details, the paper comments a few experimental results.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Non-linear dimensionality reduction; Geodesic distance; Graph distance; Spanning tree; Graph cut

*Corresponding author. Tel.: +32 27 64 47 78; fax: +32 10 47 25 98.
E-mail addresses: lee@dice.ucl.ac.be (J.A. Lee), verleysen@dice.ucl.ac.be (M. Verleysen).
[1]M.V. works as a senior research associate of the Belgian FNRS.

0925-2312/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2004.11.042

1. Introduction

The standard task in nonlinear dimensionality reduction (NLDR, see [18]) consists in taking a set of data points sampled from a smooth, low-dimensional submanifold (e.g. P-dimensional) of a high-dimensional Euclidean space (e.g. D-dimensional), and to re-embed those points in a low-dimensional Euclidean space (e.g. P'-dimensional) while preserving the local structure of the submanifold. The purpose is to find useful new coordinates for the data. In general $D \gg P' \geqslant P$. If the original submanifold has the topology of a convex region in the P-dimensional Euclidean space, then one can hope for $P' = P$, and this is the best possible situation: the new coordinates parameterize the original submanifold. However, if the submanifold has a more complicated topology, perhaps having 'holes' or 'essential loops', then sometimes P' must be chosen greater than P because it is not possible or not as easy to find a P-dimensional embedding without tearing the manifold. For example, a torus is 2-dimensional but cannot be embedded in the 2-dimensional Euclidean plane.

This paper describes a procedure for 'tearing' or 'cutting' data manifolds, modifying them so that they have no essential loops anymore. Once this is done, in many cases one can then run NLDR algorithms successfully on the modified data, whereas the same algorithms might have failed or been less useful in the initial setting.

The proposed tearing procedure is actually applied to the 'neighborhood graph' which is often constructed in recent NLDR algorithms. Edges of the neighborhood graph connect data points which are close to each other in the high-dimensional Euclidean space. For example, each data point can be connected to its K-closest neighbors (K-ary neighborhoods) or to all other points lying no further than a certain distance ε from it (ε-neighborhood). NLDR algorithms using such a neighborhood graph are for instance LLE [15,17] and related techniques [7,2], or Isomap [21] and other algorithms using geodesic distances (e.g. geodesic versions of Sammon's nonlinear mapping (NLM) and curvilinear distance analysis [13]).

The structure of the paper is as follows. After this introduction, Section 2 briefly recalls some definitions about manifolds, graphs and shows how these two concepts are put together within the framework of NLDR. In particular, the last part of Section 2 explains how essential loops of manifolds are represented in their associated graph. Next, the tearing procedure itself is described in Section 3, whereas Section 4 shows a few experimental results on artificial data. Finally, Section 5 draws the conclusions and outlines perspectives for future work.

2. Manifolds, graphs and their relationship in NLDR

2.1. Manifolds and non-contractible loops

This section briefly defines some concepts about manifolds and graphs that are used further below.

Manifolds are central objects in geometrical topology. Actually, a manifold $\mathscr{M}$ is a topological space which is locally Euclidean (i.e., around every point, there is a neighborhood which is topologically the same as the open unit ball). Although they are objects of their own right, manifolds are often represented in a certain space in such a way that its topology (or connectivity) is preserved; such a representation is called an *embedding*. In a specific embedding, coordinates of a manifold can be given by some parametric equations. If those are infinitely differentiable, then the manifold is said to be *smooth*. Independently of any embedding, the intrinsic dimension P of $\mathscr{M}$ is the number of 'free parameters' of $\mathscr{M}$, whereas $D \geqslant P$ is its embedding dimensionality. A manifold having intrinsic dimensionality P is called a P-manifold. A Q-dimensional *submanifold* of a P-manifold $\mathscr{M}$ ($Q < P$) is a subset of $\mathscr{M}$. For instance, a circle may be a smooth 1-submanifold of a torus, which is in turn a smooth 2-submanifold of the Euclidean space $\mathbb{R}^3$. In that example, both the circle and the torus are embedded in $\mathbb{R}^3$. As a particular case, any compact smooth 1-submanifold of a manifold $\mathscr{M}$ (a deformed circle drawn on the $\mathscr{M}$) is called a *loop*.

As already stated above, the goal of NLDR is to re-embed a given smooth P-manifold. Starting from an initial D-dimensional embedding, a new P'-dimensional embedding is determined, where P' is as close as possible to P. In addition to smoothness, many NLDR techniques require that the manifold to re-embed has the same topology as a convex region of $\mathbb{R}^P$. A curved sheet of paper, like the well-known Swiss roll [21], fulfills these requirements. On the other hand, a circle or a torus do not. This is because a circle or a torus are 'circular', i.e. are connected but not simply connected. Formally, a smooth manifold is said to be *simply-connected* if every loop is contractible (every compact smooth 1-submanifold can be shrunk to a single point). Connected manifolds like a circle or a torus are said to be multiply connected. The presence of non-contractible loops in a manifold is due to holes, in the topological sense of the word. For example, loops drawn around the 'donut hole' of a torus are non-contractible.

The presence of non-contractible loops in a smooth P-manifold makes its embedding more difficult, mainly because it may happen that more than P dimensions are needed to preserve its topological structure. The circle is the most typical example: at least 2 dimensions are needed to embed this 1-manifold. However, a P-manifold with non-contractible loops can still be embedded in $\mathbb{R}^P$ if one 'destroys' a well-chosen part of its connectivity. In the case of the circle, the idea of 'cutting' (or 'tearing') it at some point seems natural: the result is a manifold that is still connected (it looks like a curved line segment) but can now be embedded in $\mathbb{R}$. Following the same idea, a torus can be embedded in $\mathbb{R}^2$, but the way to tear it is not as obvious as for the circle. A last interesting example is the cylinder; this 2-manifold has non-contractible loops but can still be embedded in $\mathbb{R}^2$ by deforming it to an annulus (a disc with a hole). Nevertheless, many NLDR techniques achieve such a deformation with great difficulty; on the other hand, if the cylinder is cut along its height, one gets a manifold looking like a curved sheet of paper whose embedding in a plane is straightforward for many algorithms.

All the above arguments raise some interest in designing a procedure that can automatically cut smooth manifolds with non-contractible loops, in order to obtain

a simply-connected manifold. Unfortunately, such a procedure would be almost useless, because usual NLDR techniques use smooth manifolds only as a modeling tool. In practice, indeed, they process (noisy) points sampled from an unknown manifold. Recent NLDR techniques like Isomap or LLE, however, use the available manifold points to build a discrete representation of the manifold. Most often, this representation is a graph obtained by connecting neighboring points. This is actually an advantage because tearing a graph proves much easier than tearing a smooth manifold.

2.2. Graph representation of smooth manifolds

A graph $G = (V, E)$ consists of two sets: V is a set of vertices and E is a set of edges, which are pairs of vertices. If those pairs are (un)ordered, then the graph is said to be (un)directed. If edges may not be repeated in E, the graph is said to be simple. Usually, NLDR techniques represent manifolds by simple undirected graphs. Given a set $\mathbf{Y} = \{\ldots, \mathbf{y}_i, \ldots, \mathbf{y}_j, \ldots\}_{1 \leqslant i,j \leqslant N}$ of N points sampled from a smooth P-manifold $\mathscr{M}$ embedded in D-dimensional space, a graph G is built by associating a vertex v_i with each available point $\mathbf{y}_i$ and an edge with each pair of vertices whose associated data points are neighbors. Most often, these neighborhood relationships are determined by computing the K closest neighbors of each point (K-ary neighborhoods) or by looking at all points lying inside an ε-ball centered on each point (ε-neighborhoods). Finally, edges are often labeled with their Euclidean length, i.e. the Euclidean distance between the two data points associated with its two vertices. Such an edge-weighted graph is said to be Euclidean and is useful to approximate geodesic distances [21] with Dijkstra's algorithm [6]. A Euclidean graph can be seen as a discrete approximation of a smooth manifold. The quality of this approximation is assessed in [3] (from the point of view of geodesic distances). In a few words, the approximation holds if data points are numerous enough, not (too) noisy, and if the parameters K or ε are carefully adjusted.

In the same way as a manifold, a graph is a topological object (its topological properties are encoded in the edges). Graphs and manifolds share many common properties (they can be connected/disconnected, etc.). One of the most important ones is that graphs can be embedded in a Euclidean space, more or less in the same way as manifolds. Graph embedding [5] is a problem somewhat related to NLDR. To some extent, Isomap or LLE can be seen as graph embedding techniques that starts by transforming the coordinates of data points into a Euclidean graph.

What does happen when a smooth manifold with non-contractible loops is represented by a Euclidean graph G? Actually, if the quality of the graph approximation is good, non-contractible loops will be replaced in the Euclidean graph with 'non-contractible' cycles. In a graph G, a smooth l-submanifold is somewhat equivalent to a path, i.e. a subgraph that can be encoded as a sequence of vertex such that an edge connects two successive vertices ($\pi = [\ldots, v_i, v_j, v_k, \ldots]$ and $\{v_i, v_j\}, \{v_j, v_k\} \in E$). Then, a loop (compact smooth 1-submanifold) corresponds to a

'circular' path, in the sense that the first vertex of the first edge is the same as the second vertex of the last edge. Assuming further that

- a b-cycle is a cycle having b edges,
- an elementary cycle is a cycle with no more than c edges,
- two cycles of the same graph G are t-tangent if they share at least t edges,

then any b-cycle γ can be 'continuously' deformed (i.e. modified step by step) by repeatedly running the following simple procedure:

(1) Find an elementary cycle τ that is at least 1-tangent to γ.
(2) Remove from γ and τ all edges they share.
(3) Replace the missing edges in γ with the remaining edges of τ.

At each execution, this procedure replace an b-cycle γ with a $(b \pm d)$-cycle $(0 \leqslant d < c)$ which is at least $(b - c + 1)$-tangent to γ. Non-contractible cycles are detected in a graph when there are cycles that cannot be deformed to elementary cycles using the above procedure.

In the same way as non-contractible loops in a manifold, non-contractible cycles in a graph usually complicate its embedding in a low-dimensional space. Hence, it would be useful to tear the graph in order to cut all non-contractible cycles before embedding it.

3. Tearing graphs with non-contractible cycles

The task of detecting non-contractible loops in a smooth manifold or non-contractible cycles in a graph proves very difficult in both cases, because many possibilities should be explored. Moreover, for each manifold loop or graph cycle, testing whether it can be deformed to a single point or an elementary cycle would be very time-consuming. Finally, even if it is assumed that non-contractible loops/cycles have been detected, how to achieve the tear?

In the case of a graph, detecting non-contractible cycles and cutting them can be combined into a single and simple procedure. The key idea is the following: instead of detecting and cutting non-contractible cycles in a graph G, cut all cycles (by removing an appropriate subset of edges) and then put back edges one at a time, making sure not to generate any non-contractible cycle. These two steps are detailed in the two next subsections.

3.1. Cutting all cycles

The first stage of the tearing procedure (cutting all cycles in a graph G, see central plots in Fig. 1) can be achieved very easily by using standard algorithms that compute a spanning tree of G. By definition, a *tree* $T = (V, E_T)$ is a connected graph with no cycles. More formally, a graph is *connected* if at least one path connects

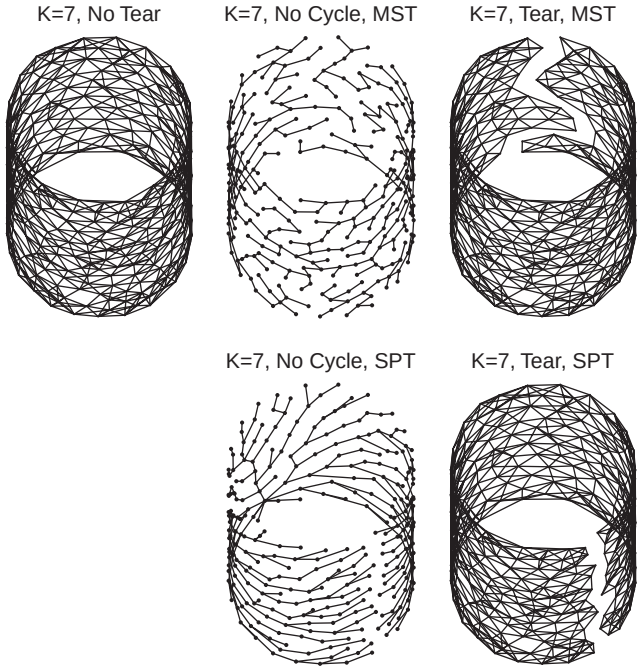


Fig. 1. The two-stage tearing procedure: (left) 300 points lying on a cylinder, connected to their 7 closest neighbors, (center) the minimum spanning tree (MST) and a shortest path tree (SPT) computed on the resulting graph, (right) the torn graph after reintroduction of all edges that do not generate non-contractible cycles with more than 4 edges.

every pair of vertices; this means that a connected graph trivially includes at least $|V|-1$ edges. As a particular case, a tree is a connected graph with precisely $|V|-1$ edges and where only a single path connects each pair of vertices; this prevents the presence of any cycle in a tree. Then, starting from a connected graph $G=(V,E)$, a *spanning tree* $G_T=(V,E_T)$ is a subgraph of G ($E_T \subset E$) that is a tree. Two well known kinds of spanning trees are minimum spanning trees (MST) and shortest path trees (SPT).

MST can be computed by either Kruskal's [9] or Prim's [14] algorithm. In a connected graph where edges are labeled with real numbers, Prim's algorithm provides a spanning tree such that the sum of its edge lengths is minimum. MST are often used to identify a minimum cost network to connect e.g. computers (with wires) or cities (with railroads). When all edges of a connected graph have different real numbers as labels, the MST is unique.

SPT are computed by Dijkstra's algorithm [6], which solves the well-known single-source shortest paths (SSSP) problem. If the algorithm is run on a connected graph whose edges are labeled with positive real numbers, and given a source vertex v_i, the algorithm computes all shortest paths from v_i to all other vertices v_j ($1 \leqslant j \leqslant N, j \neq i$).

It is then not difficult to see that the resulting shortest paths can be encoded as a tree: each vertex v_j is connected to the single following vertex on the shortest path leading to the source vertex v_i. This requires exactly $N-1$ edges since the source vertex v_i is not connected to itself. Contrarily to a MST, a SPT is not necessarily unique: there may be at most N different SPTs.

At first glance, it is not evident at all to guess whether an MST would behave better or worse than a SPT in the context of manifold tearing. This question is answered in the experimental section (Section 4).

3.2. Reintroducing discarded edges

Once a spanning tree G_T of G is computed, the second stage (Fig. 1 right) of the tearing procedure may be run. This stage aims at determining a graph $G_C = (V, E_C)$, having exactly the same vertices as the initial graph G, but with a smaller edge set $E_C \subset E$, such that G_C has no non-contractible cycles. For this purpose, edges discarded during the first stage are gathered in a set $S = E \backslash E_T$ and G_C is initialized with G_T. Then edges in S are considered one at a time, in an adequate order, by means of a breadth-first traversal of G along its spanning tree G_T. Because G_C is initialized with G_T, adding an edge coming from S to E_C always generates one or more cycles in G_C. At this point, the important step of the tearing procedure consists in checking that the addition of an edge generates only contractible cycle. Practically, this is done by computing the number of edges of the shortest cycle generated by the addition of an edge and testing that this number is less or equal to c, the maximum number of edges in an elementary cycle. An easy way to perform this comparison, before adding the edge, is to compute in G_C the shortest path between both vertices of the considered edge: if this number is smaller than c, then the edge may be added to G_C. All these ideas are detailed more formally in the pseudo-code listed in Fig. 2.

Intuitively, the algorithm detailed in Fig. 2 progressively 'welds' the branches of the tree G_T together, starting from the root v_r of the tree and using edges available in S; it stops when it reaches the leaves of the tree. Because the algorithm explores G_T in a breadth-first way, cycles generated by the addition of an edge can only occur in the region of G_C that has already been explored (removing the explored vertices and associated edges leaves a forest of disconnected trees). This ensures that edges are considered in the right order.

The important point in the above algorithm is the computation of the path from v_i to vertex v_j having the smallest number of edges (call to the function MIN_NUMBER_EDGES( ) in Fig. 2). Fortunately, this task can be performed very efficiently. Indeed, instead of using Dijkstra's algorithm, a simplified search procedure has been designed, based on the following observations:

- The problem is symmetric with respect to v_i and v_j (there can be two source vertices, whereas the problem solved by Dijkstra's algorithm (SSSP) has only one.
- Edges lengths may be neglected (all edges have unit weight).
- The search procedure may be stopped as soon as it is sure that the shortest path between v_i and v_j includes more than $c-1$ edges.

```
TEAR_STAGE2(G, G_T, c)
Input: G = (V, E) (original Euclidean graph),
       G_T = (V, E_T) (spanning tree of G, with root vertex v_r),
       c (the maximum number of edges in an elementary cycle).
Output: G_C = (V, E_C) (torn G i.e. subgraph of G without any non-contractible cycle including more than c edges).
Auxiliary: Q (a queue data structure), v_i, v_j (vertices), l (integer).
Begin
  ▷ Initialize G_C to G_T.
  G_C ← G_T
  ▷ Breadth-first traversal of the spanning tree of G.
  Q ← [v_r]
  while Q is not empty do
    ▷ Extract top element of the queue Q.
    v_i ← Q.POP()
    ▷ Visit each vertex v_j directly connected to v_i.
    for each neighbor v_j of v_i ∈ V do
      if edge {v_i, v_j} ∈ E_T then
        ▷ Put vertex v_j at the end of the queue.
        Q ← [Q, v_j]
      else
        ▷ Check that edge {v_i, v_j} may be introduced in G_C,
        ▷ i.e. compute in G_C the path from vertex v_i to vertex v_j
        ▷ having the smallest number l of edges (edge lengths are neglected).
        l ← G_T.MIN_NUMBER_EDGES(i, j, c)
        if l < c then
          ▷ The smallest cycle generated by the addition the edge {v_i, v_j} to G_C is either elementary,
          ▷ meaning that no non-contractible cycle is generated: it may be added to G_C.
          E_C ← E_C ∪ {{v_i, v_j}}
        end if
      end if
    end for
  end while
  Return G_C
End
```

Fig. 2. Pseudo-code for the second stage of the tearing procedure (comment lines begin with a triangle).

Hence, combining these ideas allows writing function MIN_NUMBER_EDGES( ) as two symmetric breadth-first traversals of the graph G_C (see pseudo-code in Fig. 3).

The symmetric traversals combined with the early stopping conditions make the procedure very fast. For example, suppose G_C is a Euclidean graph that is embedded in $\mathbb{R}^2$; if the shortest path between v_i and v_j includes $l < l_{\max}$ edges, the value of l is determined in $\mathcal{O}(2\pi\lceil l/2\rceil^2)$ (explored vertices lie in two growing circles centered on v_i and v_j; the symmetric traversals stop when both circles are tangent to each other). This means that checking the shortest path between two vertices is fast on average. Obviously, the true complexity of the algorithm depends on several factors (number of vertices, minimum embedding dimension of the Euclidean graph, number of edges per vertex, shape of the graph, etc). and is difficult to evaluate. As an example, the

Fig. 3. Pseudo-code for a fast function computing the path having the smallest number of edges in a graph G (comment lines begin with a triangle).

```
MIN_NUMBER_EDGES(i, j, l_max)
Input: G = (E, V) (a graph), i, j (the indices of two vertices of G), l_max (an upper bound of the number of edges).
Output: the number l of edges of the path between v_i and v_j having the smallest number of edges;
        the value of l is either less or equal to l_max (a path with no more than l_max edges exists)
        or ∞ (there is no path with no more than l_max edges).
Auxiliary: v, v_k, v_l, v_m, v_n (vertices from V, Q_1, Q_2 (queue data structures).
Begin
  ▷ Initialize each vertex as being unvisited.
  for each vertex v ∈ V do
    v.distance ← −1
    v.mark ← 0
  end for
  ▷ Initialize two queues with both sources vertices v_i and v_j.
  Q_1 ← [v_i]
  Q_2 ← [v_j]
  ▷ Mark both sources.
  v_i.distance ← 0
  v_i.mark ← 1
  v_j.distance ← 0
  v_j.mark ← 2
  ▷ Symmetric breadth-first traversals of G.
  while neither Q_1 nor Q_2 are empty do
    ▷ Extract the top elements of both queues.
    v_k ← Q_1.POP()
    v_l ← Q_2.POP()
    ▷ Check that the threshold c is not reached.
    if v_k.distance + v_l.distance ≥ l_max then
      l ← ∞
      Return l
    end if
    ▷ Visit each vertex v_m directly connected to v_k
    for each neighbor v_m of v_k do
      if v_m.mark = 2 then
        ▷ A path is found! Return its length.
        l ← v_k.distance + 1 + v_m.distance
        Return l
      else
        ▷ Push v_m in the queue and mark it as visited by Q_1.
        Q_1 ← [Q_1, v_m]
        v_m.distance ← v_k.distance + 1
        v_m.mark ← 1
      end if
    end for
    ▷ Visit each vertex v_n directly connected to v_l.
    for each neighbor v_n of v_l do
      if v_n.mark = 1 then
        ▷ A path is found! Return its length.
        l ← v_l.distance + 1 + v_n.distance
        Return l
      else
        ▷ Push v_n in the queue and mark it as visited by Q_2.
        Q_2 ← [Q_1, v_m]
        v_n.distance ← v_l.distance + 1
        v_n.mark ← 2
      end if
    end for
  end while
End
```

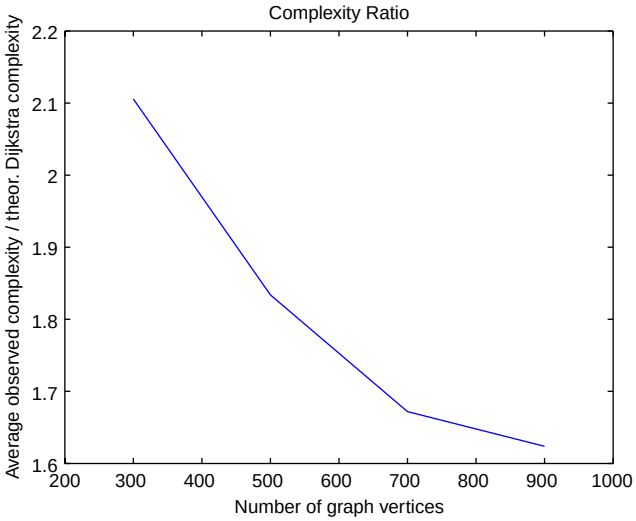


Fig. 4. The 'rice' image database: performance measure. The total number of vertices visited by the function MIN_NUMBER_EDGES is divided by the complexity of Dijkstra's algorithm (used to build the SPT, $\mathcal{O}(N\log_2 N)$). The cost has been measured for 4 different numbers of vertices in the graph G ($N = 300, 500, 700, 900$) and averaged over the N possible SPTs. Data points are intrinsically two-dimensional and G is built using K-ary neighborhoods ($K = 6$). (Note: Please refer to the web-version of the paper to see the colour version of the figure.)

practical complexity of the tearing procedure has been measured in the case of the 'rice' image database, which is studied in Section 4. The tearing procedure has been run on graphs having different numbers of vertices, as shown in Fig. 4. In that experiment, the manifold has two free parameters, each point is connected to its 6 closest neighbors, $c = 5$ and the tearing procedure uses SPTs computed by Dijkstra's algorithm. The total number of vertices that are visited by the function MIN_NUMBER_EDGES( ) during a run of the tearing procedure is compared with the complexity of Dijkstra's algorithm ($\mathcal{O}(N \log_2 N)$) and displayed as a ratio in Fig. 4. For each value of the number of vertices ($N = 300, 500, 700, 900$), the displayed costs are averaged over the N possible SPTs and divided by the corresponding complexity value of Dijkstra's algorithm. As can be seen, the cost of the whole tearing procedure is more important than the one of Dijkstra's algorithm, but its relative importance decreases as the number of vertices grows. Anyway, this cost is negligible in case Isomap is run afterwards, because the latter requires to run Dijkstra's algorithm N times, in order to approximate all pairwise geodesic distances from each vertex.

3.3. Optimization of the tear

It is easy to see that the proposed tearing procedure is deterministic, i.e. yields a unique graph G_C, for a given value of its parameters G, G_T and c. Considering that G depends on the available data, the user chooses the value of c and the type of spanning tree (MST or SPT). As already stated above, G has a unique MST if all

edge lengths are different; with randomly distributed data, this condition holds most of the time. Hence, in the case of a MST, the tearing procedure yields a unique result.

On the other hand, a graph G with N vertices has generally several SPTs (at most N). When running Dijkstra's algorithm, the obtained SPT depends on the source vertex that is specified. Quite naturally, the source vertex can be seen as a free parameter that can be optimized. Possible criterions to be optimized are for example the number of torn edges or their summed length; the last proposition seems more natural in the case of a Euclidean graph. Those numbers can be either minimized or maximized by simulating successively the tear obtained with each possible SPT. Afterwards, the tear that reaches the best value of the criterion is chosen and truly achieved.

From the computational point of view, the above iterative optimization requires to run Dijkstra's algorithm N times to find the N possible SPTs. As observed in the previous section, the cost of the tearing procedure may be assumed similar to the one of Dijkstra's algorithm. Within the framework of NLDR, e.g. using Isomap, that cost remains reasonable.

3.4. Related algorithms

Graph cutting [19] is a central problem in graph theory and has many applications, for instance in chip and circuit design, reliability of communication networks, etc. Many algorithms solve the following fundamental problem: find the minimum cut of an undirected edge-weighted graph. More precisely, it consists in finding a non-trivial partition of the graph vertex set V into two parts such that the cut weight (the sum of the weights of the edges connecting the two parts) is minimum. This goal looks very similar to the one presented in the previous subsection. In spite of this resemblance, min-cut algorithms perform a task that is totally different from what the above-described tearing procedure does. Indeed, min-cut algorithms compute a graph partition, whereas the tearing procedure yields a graph that remains connected.

4. Experimental results

This section shows some results of the proposed tearing procedure on simple manifolds, which are artificially generated. As the aim is to illustrate visually how the tearing procedure works, most of them are two-dimensional manifolds embedded in a $\mathbb{R}^3$: a cylinder, a torus and a holed cylinder. It is noteworthy that representative points of the manifolds are obtained as advised in [11,13]: a large number of points (several thousands) are randomly drawn and processed by a simple vector quantization method (Competitive Learning [1]). This yields a small subset of points which are regularly spaced on the manifold.

Several NLDR methods are used to embed the proposed data manifolds in a plane. These methods are Isomap [20,21], Sammon's NLM [16] and curvilinear component analysis [4,8] (CCA). These three methods try to find an isometry between the available high-dimensional data points and their representation in a

low-dimensional space. In order to compare with Isomap, geodesic versions of NLM and CCA, namely GNLM and GCCA [11,13], are used. As in Isomap, distances measured in the high-dimensional space are geodesic ones, whereas usual Euclidean distances are computed in the low-dimensional space.

Two additional NLDR methods, namely Isotop [12,10] and LLE [15], which are based on topology preservation, are also used. To the authors' knowledge, CCA is the only existing NLDR method that is intrinsically able to tear a manifold, without any preprocessing.

For simple and illustrative examples, like the cylinder or the torus, only one method is used (GNLM). For the holed cylinder, the four methods are compared in order to show how each of them reacts to different types of tears.

4.1. Cylinder

Fig. 5 clearly shows the advantage of tearing a manifold with non-contractible loops (a cylinder) before reducing its dimensionality. If the manifold is not torn, its different

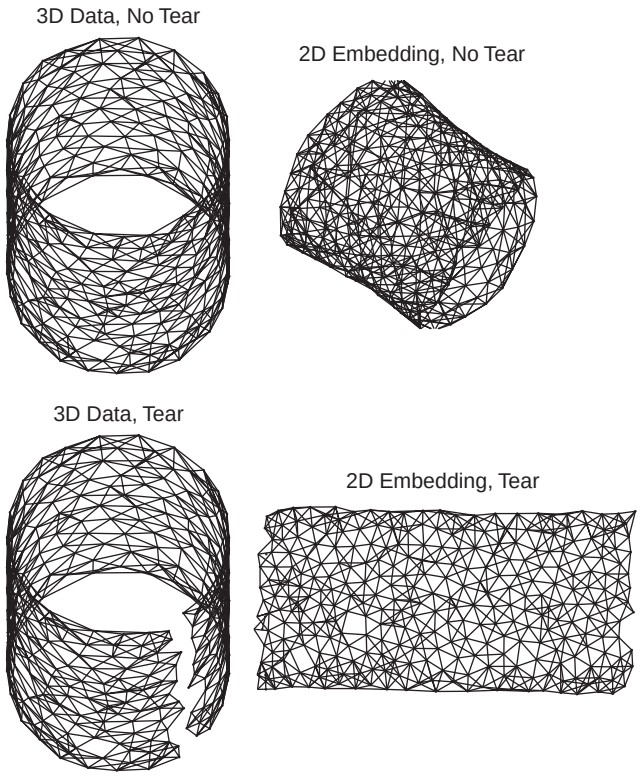


Fig. 5. Two-dimensional embedding of the same cylinder as in Fig. 1 using Sammon's nonlinear mapping with the graph distance: (top) the 300-vertex Euclidean graph is embedded without tearing any edges, (bottom) embedding obtained when the same graph is torn (using a shortest path tree and $c = 4$).

regions may get deformed or superimposed in the low-dimensional embedding space (second plot in Fig. 5). On the other hand, after running the tearing procedure, it only needs to be 'unrolled' in order to get a good embedding (fourth plot in Fig. 5).

As advised in [11], the 300 available points of the cylinder are obtained by performing a vector quantization on a larger distribution (10 000 points, without noise). In order to build the graph, each point is connected to its 7 closest neighbors. Next, graph distances are computed using Dijkstra's algorithm and processed by GNLM, which yields the first embedding. As can be seen, the cylinder is crushed or flattened instead of being unrolled.

If the graph is torn before, in order to suppress all non-contractible cycles ($c = 4$), GNLM yields the desired embedding. The tear is obtained using a SPT, because it produces a straighter and neater cut than the MST (see Fig. 1).

4.2. Torus

A slightly more difficult example is the torus, shown in top of Fig. 6. As for the other manifolds, the 600 available points are obtained by performing a vector quantization on a larger distribution (30 000 points, without noise). Each point is connected to its 6 closest neighbors.

Because tearing a torus is not an obvious task, this is an ideal manifold to illustrate how NLDR techniques behave with or without a preliminary run of the proposed tearing procedure. For this purpose, two-dimensional embeddings are computed by GCCA, which is the only NLDR algorithm having the intrinsic ability to tear manifolds, as mentioned in the beginning of Section 4. However, when GCCA has to stretch and tear a manifold, it usually converges slowly; the goal is then to see whether the use of the tearing procedure can help GCCA to converge. Five embeddings are computed and displayed in Fig. 6. Their quality is assessed by evaluating Sammon's stress [16] (this criterion is better than the one proposed in [4,8], which depends on a lot of parameters).

As was already visible in Fig. 1, the shape of the tear mainly depends on the chosen kind of spanning tree. The MST computed by Prim's algorithm unfortunately leads to a very long and meandering tear. Yet, the idea of a minimum-weight spanning tree was appealing: the tree retains the shortest edges, i.e. the intuitively 'strongest' ones. On the other hand, a SPT computed by Dijkstra's algorithm is not unique (it depends on the source vertex), but leads to a more regular tear. If the source vertex is chosen in order to minimize or maximize the summed lengths of the torn edges, very nice results are obtained. In the former case, the torus is torn along its smallest horizontal circle; in the latter case, the tear follows the largest horizontal circle.

Not surprisingly, Sammon's stress reaches the lowest value for the minimum tear computed with a SPT.

4.3. Holed cylinder

The first plot of Fig. 7 shows a cylinder very similar to the one of Fig. 5, except that two lateral holes are digged. The diameter of the cylinder is 1.0 and the one of

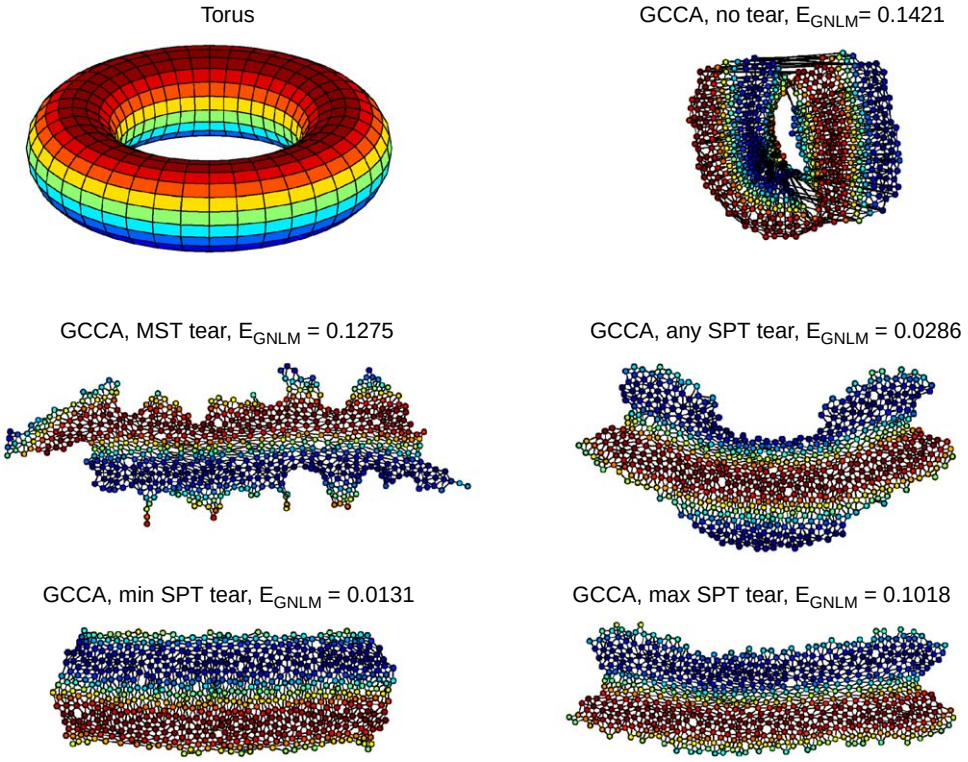


Fig. 6. Two-dimensional embedding of a torus using GCCA: (top left) the torus manifold, where the color varies according to the height, (top right) embedding of 600 points, the graph is not torn beforehand, (middle left) embedding of the same data set after breaking all non-contractible cycles ($c = 4$) by means of the proposed tearing procedure, using an MST, (middle right) idem, but with an SPT, chosen randomly among the 600 possible ones (bottom left) idem, with the SPT that leads to the shortest cut, (bottom right) the same again, but with the SPT that leads to the longest cut. (Note: Please refer to the web-version of the paper to see the colour version of the figure.)

the side holes is 0.5. The 800 available points are obtained by performing a vector quantization on a larger distribution (17 000 points, without noise). Each point is connected to its 7 closest neighbors. The other plots of Fig. 7 shows how four different methods embed the data set in a plane, without any prior tear. These methods are Isomap, GNLM, GCCA and Isotop. As can be seen, Isomap and the GNLM perform poorly: the cylinder is crushed like an empty can and large regions of the manifold get superimposed. GCCA yields a much better result, with no superimposition, thanks to its intrinsic ability to tear the manifold. As Isomap and GNLM, Isotop cannot tear the manifold but still manages to minimize the superimposition; because topology preservation is a milder condition than distance preservation, Isotop achieves such result by distorting the manifold. How do those four methods behave if the tearing procedure preprocess data? Before answering this question, it is noteworthy that the holed cylinder has three holes (i.e. different

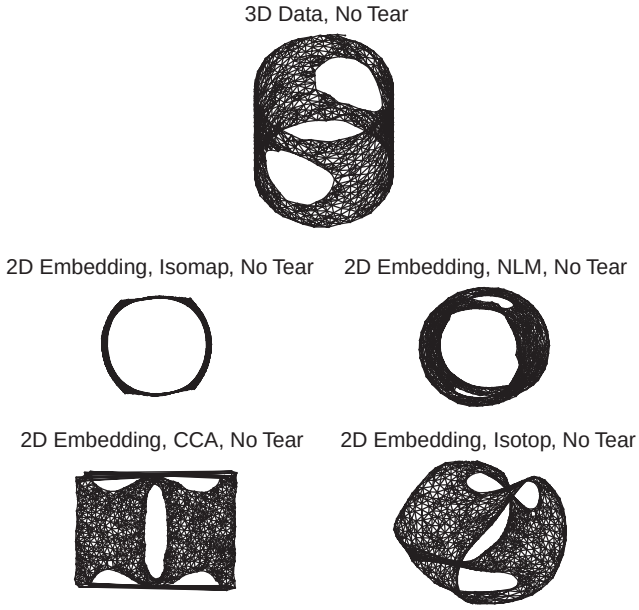


Fig. 7. Holed cylinder: three-dimensional data and two-dimensional embeddings achieved four NLDR algorithms (Isomap, CCA, Sammon's NLM and Isotop).

families of non-contractible loops). However, side holes must not be considered on the same footing as the main hole. Indeed, non-contractible loops around the side holes need not to be broken in order to embed the holed cylinder. Indeed, the holed cylinder could be torn in exactly the same way as the one in Fig. 5, i.e. just as if the two lateral holes did not exist. Actually, breaking the non-contractible loops related to the side holes would even be wasteful since doing so uselessly destroy part of the manifold connectivity.

Figs. 8–11 show the embeddings computed by Isomap, GNLM, GCCA and Isotop, respectively. The tearing procedure is run with different parameter settings:

- Two different maximum numbers of edges for the allowed non-contractible cycles are proposed ($c = 5$ or $c = 30$).
- Two different spanning trees are used (MST or SPT).
- In the case of the SPT, the source vertex given to Dijkstra's algorithm is either randomly chosen or the one leading to the maximum and minimum tears.

If $c = 5$, all non-contractible cycles get broken, but a higher value ($c = 30$) allows considering cycles around the lateral holes as elementary cycle: only the longer cycles associated with the main hole are considered as non-contractible. The different types of tears (MST, any SPT, min SPT, max SPT) yield the expected results. As in previous experiments, the MST leads to a long and meandering tear, whereas SPTs yields straighter cuts.

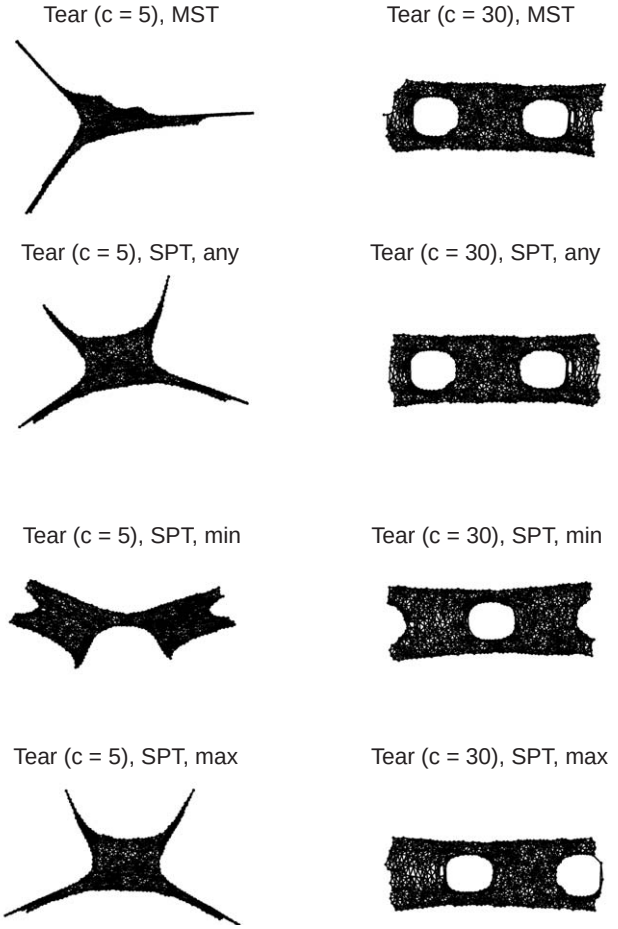


Fig. 8. Two-dimensional embeddings of the holed cylinder: results of Isomap after tearing the manifold with different parameters.

The results of Isomap (Fig. 8) are dramatically improved when the holed cylinder is preprocessed by the tearing procedure. The performances of Isomap are very sensitive to the convexity of the manifold; holes, for example, decrease the quality of the embedding [13]. If $c = 5$, lateral holes are torn and some regions of the manifold becomes loosely connected to other ones, like peninsulas (they are more visible on the left of Fig. 10). That configuration harms the hypothesis of convexity even more than holes, explaining why Isomap tends to stretch those regions (embeddings have spiky corners). Things get much better when $c = 30$: convexity still does not hold, but connectivity is better, what prevents Isomap to distort the manifold.

The results of GNLM (Fig. 9) fairly compare to those of Isomap. Instead of distorting loosely connected regions, GNLM sometimes twists the manifold. Again, this happens because the holed cylinder is not a convex manifold: a perfect isometry

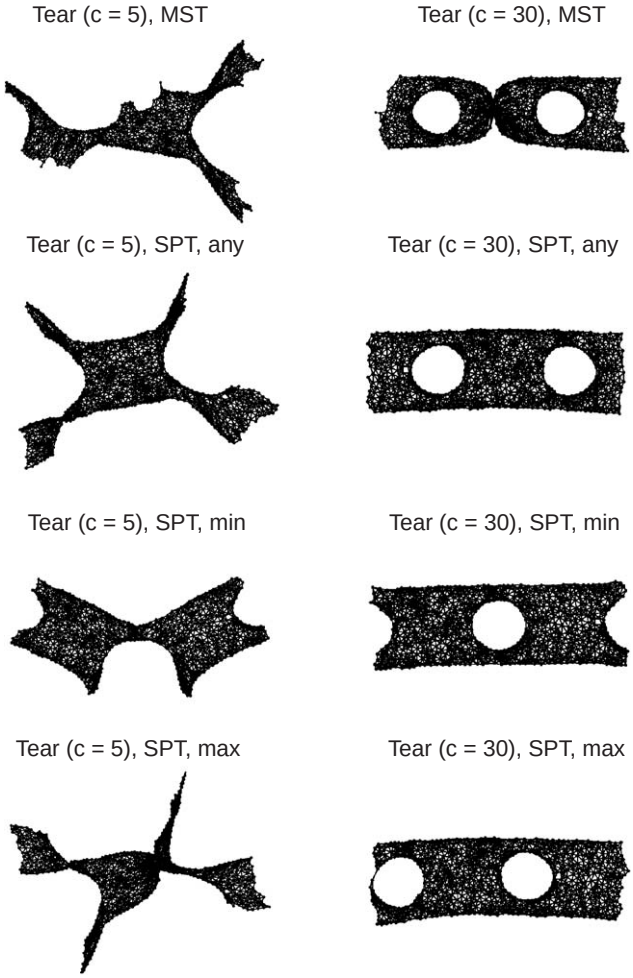


Fig. 9. Two-dimensional embeddings of the holed cylinder: results of the GNLM after tearing the manifold with different parameters.

between geodesic distances and Euclidean ones does not exist. Indeed, because of the two side holes, some geodesic distances get overestimated (the corresponding shortest path has to circumvent the holes, whereas Euclidean distance may fly over them). In this case, a twisted embedding allows stretching the Euclidean distances corresponding to overestimated geodesic ones.

Fig. 10 shows that GCCA outperforms both Isomap and the GNLM. Minor twists are observed when $c = 5$ but distortions are nearly absent. These good results may be explained by the fact that GCCA attempts to preserve short distances prior to long ones [4,8]. This trick allows neglecting long graph distances that are overestimated because of holes.

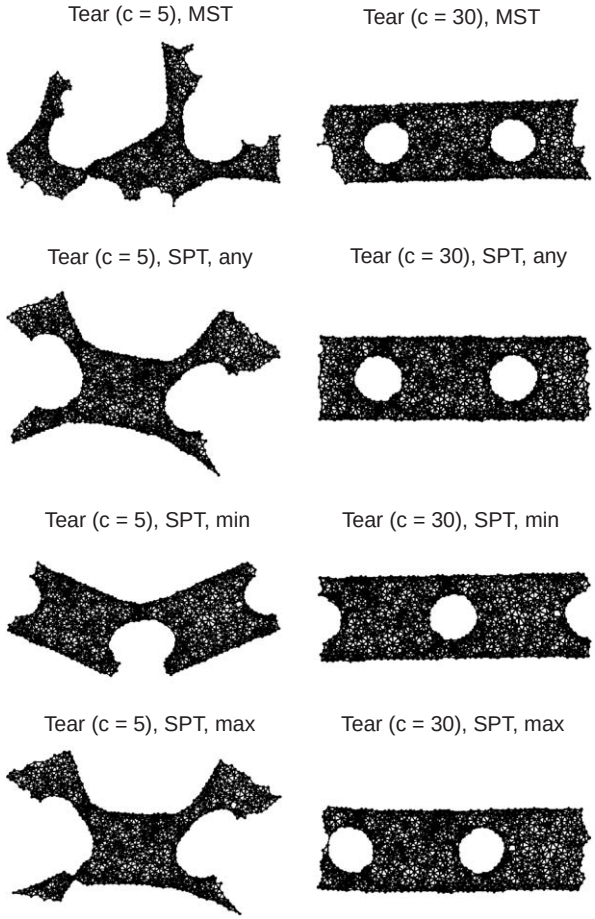


Fig. 10. Two-dimensional embeddings of the holed cylinder: results of GCCA after tearing the manifold with different parameters.

Results of Isotop are slightly different from the three previous NLDR algorithms, because it is based on topology preservation rather than on distance preservation. This explains the 'slack and spongy' look of embeddings displayed in Fig. 11.

The example of the holed cylinder shows that tearing a manifold with non-contractible loops helps to reduce its dimensionality. As it could be expected, the best results are attained with the minimum-length tear (which is often the most regular one too).

4.4. The 'rice' image database

As already proposed in other papers (e.g. [21,15]), NLDR can be used in image processing, in order to sort huge set of images. In this example, 10 000 images of a

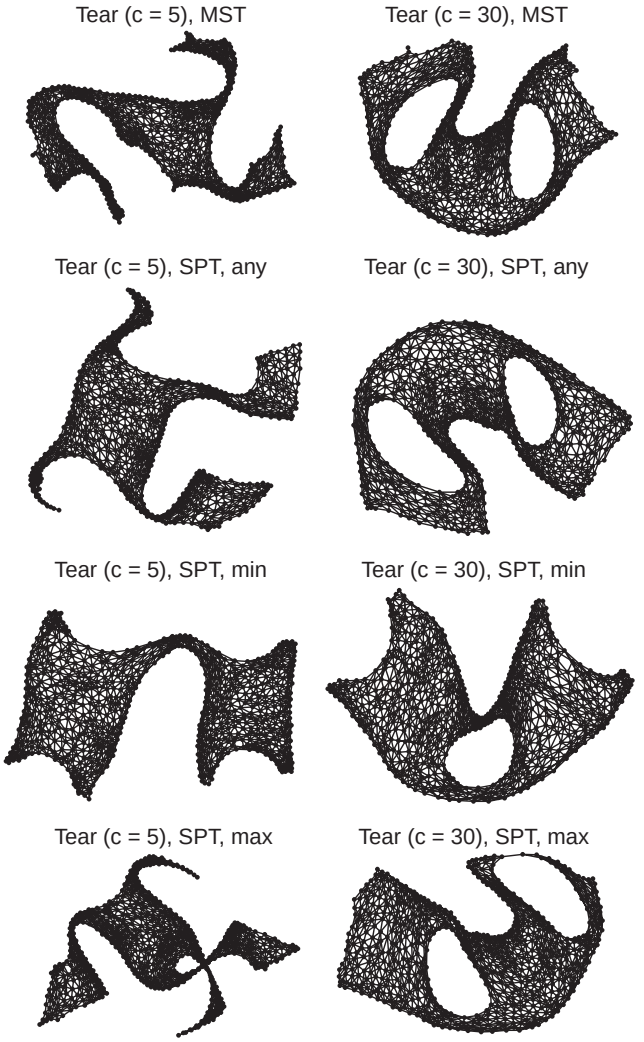


Fig. 11. Two-dimensional embeddings of the holed cylinder: results of Isotop after tearing the manifold with different parameters.

rice seed have been artificially generated. Each image consists of a 16×16 array of gray-level pixels; the background is black and the rice seed, drawn as a stretched two-dimensional bell, is white with shades of gray. A small subset of the database is displayed in Fig. 12. Just as for previous examples, the database has been preprocessed by a vector quantizer, in order to retain a small but representative subset of the whole database.

There are two free parameters in the database: the rotation angle of the seed and its left–right position. Hence, theoretically, NLDR could be applied to embed the

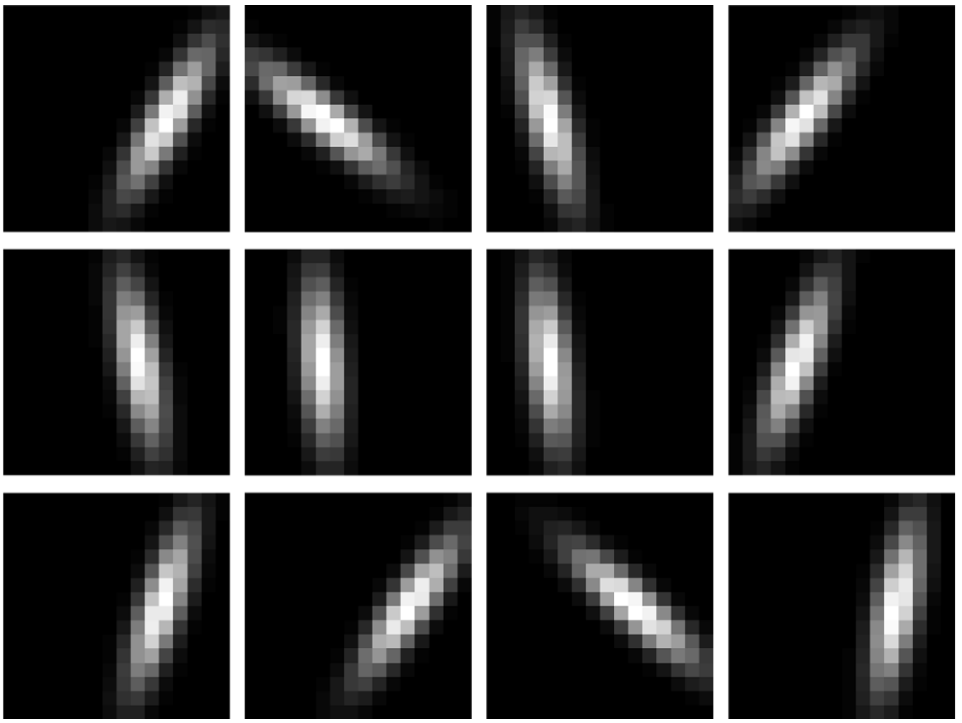
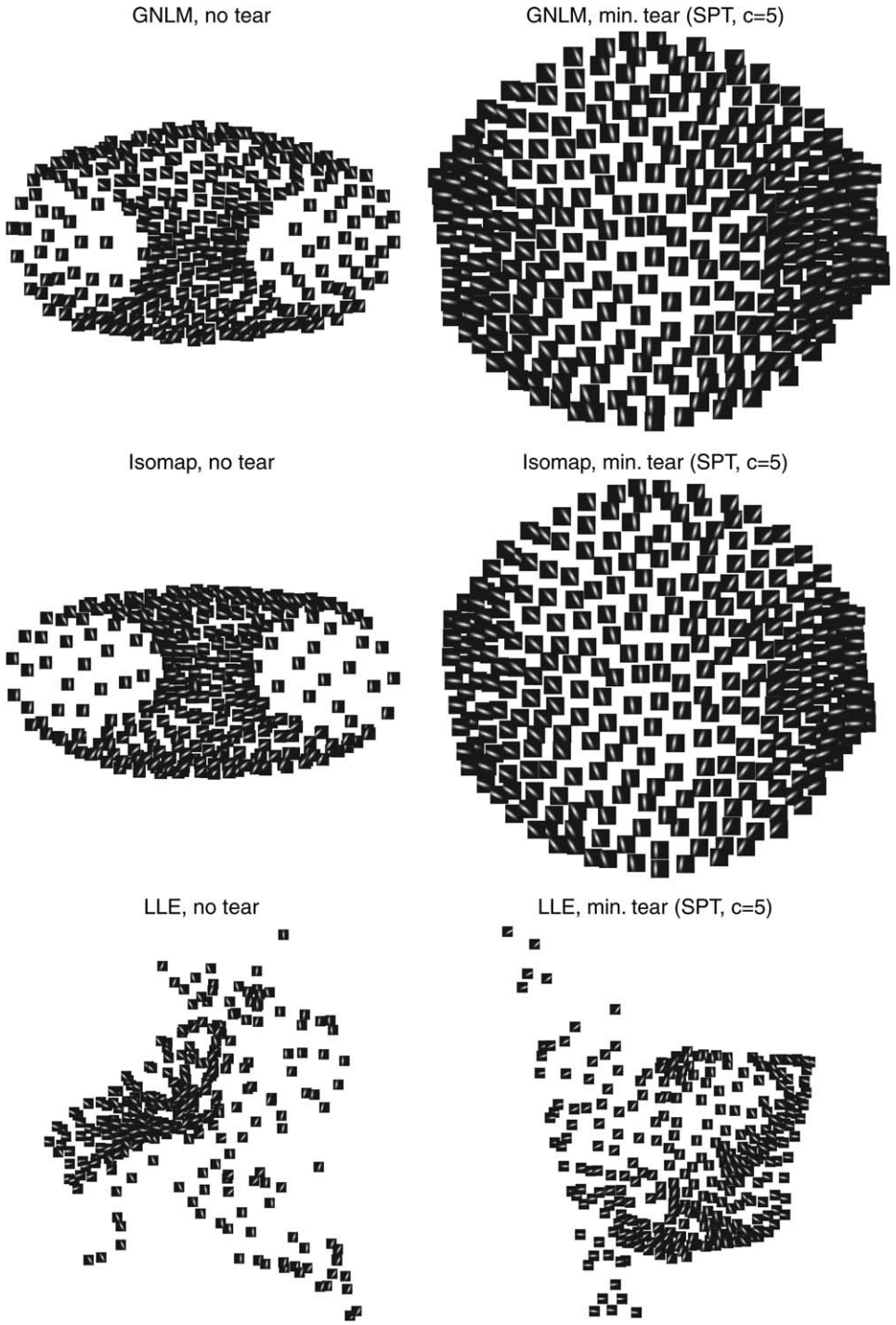


Fig. 12. The 'rice' image database: a few images taken from the database; each image is an array of 16×16 gray-level pixels.

data in a plane. For this purpose, three NLDR algorithms are used: Isomap, GNLM and LLE. These algorithms process the raw images that are output by the vector quantizer (256-dimensional vectors). In the case of this example, however, embedding the images in a plane may prove a difficult task, because the first free parameter is a rotation angle, meaning that the manifold is 'circular' with respect to this parameter. Results of the three NLDR algorithms are shown in Fig.13. As expected, it can be guessed from the embeddings in the left column that the manifold contains non-contractible loops (it actually looks like a torus quarter). These embeddings are not ideal, because different regions of the manifold get superimposed. On the other hand, embeddings displayed in the right column, obtained after tearing the manifold, are much more satisfying: similar images are embedded close to each other and there are no superimpositions anymore.

Fig. 13. The 'rice' image database: two-dimensional embedding of 300 representative images (obtained with vector quantization); three NLDR algorithms are used (Isomap, GNLM and LLE) with K-ary neighborhoods ($K = 6$). In the left column, the 300 images are embedded without any preprocessing, whereas in the right column, the tearing procedure has been applied before.


GNLM, no tear
GNLM, min. tear (SPT, c=5)
Isomap, no tear
Isomap, min. tear (SPT, c=5)
LLE, no tear
LLE, min. tear (SPT, c=5)

4.5. Removal of undesired edges

Cutting non-contractible cycles in a graph may be useful even for manifolds that have no non-contractible loops. Indeed, when building a graph in order to represent an unknown data manifold, it may happen that undesire edges appear in the graph and totally jeopardize the graph representation of the underlying manifold. This may raise some major difficulties in some NLDR techniques. For instance, in Isomap, the approximation of geodesic distances by graph distances may be completely wrong because of a single undesired edge [13] that connects two different regions of the manifold.

Undesired edges typically appear when data points are noisy and connected to a too high number of neighbors, as illustrated by the spiral in Fig. 14. With three neighbors only (first plot in Fig. 14), there are no undesired edges but the approximation of the geodesic distances is very rough (the graph is sparse). Approximations would be far better with 7 neighbors (second plot in Fig. 14), but then a few undesired edges appear in the graph. Those edges are like short-circuits in an electrical system: their presence make the result of Dijkstra's shortest path algorithm completely different. How to remove those undesired edges?

In most cases, edges are undesired when they generate long non-contractible in the graph associated with a manifold that has normally no non-contractible loops. The

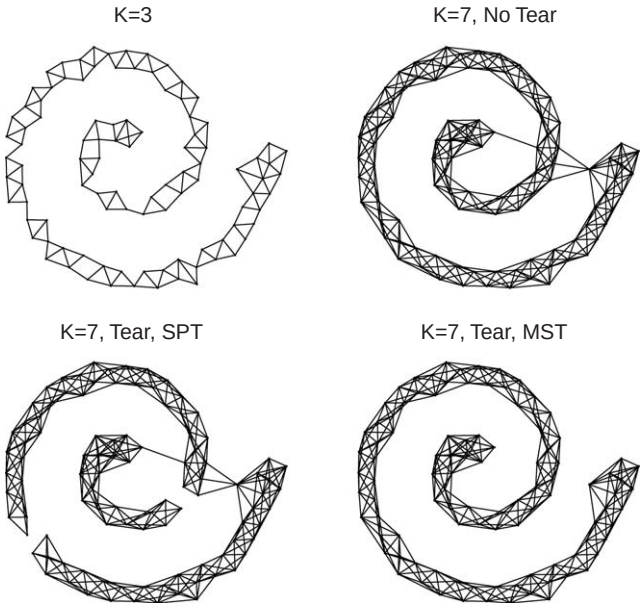


Fig. 14. Tearing undesired edges: (first plot) some noisy points of a spiral-shaped manifold, each point is connected to its 3 closest neighbors, (second plot) the same points connected to their 7 closest neighbors, undesired edges appear in the graph, (third plot) the latter graph after removing the non-contractible cycles with a SPT, (fourth plot) with a MST.

tearing procedure described in Section 3 allows removing them in an elegant way. The two last plots in Fig. 14 show the results when using, respectively, a SPT and MST.

As can be seen, the SPT computed by Dijkstra's algorithm (bold edges) does not yield the expected result. This is not very surprising, as undesired edges precisely provide Dijkstra's algorithm with opportunities to shrink some paths. As a consequence, undesired edges get almost systematically included in the SPT. This does obviously not solve the problem and leads to an even worse case.

On the other hand, using an MST in the tearing procedure yield a much better result. By construction, an MST tends to include the shortest edges and to exclude longer ones. As undesired edges typically appear in sparse regions of the data distribution, they are often longer than normal edges and should likely be eliminated when computing the MST. This explains why the tearing procedure performs better with an MST than with a SPT in this case.

5. Conclusion

This paper has presented an efficient procedure to tear manifolds with holes or non-contractible loops, within the framework of nonlinear dimensionality reduction. The procedure relies on the fact that most recent NLDR methods build a graph to represent the unknown manifold. In that case, tearing the manifold corresponds more or less to cutting all non-contractible cycles in its associated graph. Instead of directly identifying and breaking the non-contractible cycles, which would be a tedious task, all cycles are broken by computing a spanning tree and then all edges that do not generate non-contractible cycles are put back in the graph, one by one.

The proposed tearing procedure may be useful in many NLDR algorithms. Experiments are conducted with five methods: Isomap [21], Geodesic Sammon's NLM [16,11,13], Geodesic CCA [11,13], LLE [15] and Isotop [10].

Experimental results have demonstrated that the procedure can tear manifolds like cylinders or tori. The influence of different parameters of the procedure is investigated, showing how to get various types of tears (minimum or maximum length, etc.). Moreover, the same tearing procedure elegantly addresses the important issue of undesired edges in the graph representation of data manifolds. Indeed, the procedure can efficiently remove undesired edges that often jeopardize the operation of NLDR algorithms.

Yet, although the tearing procedure proves successful for 'circular' manifolds (cylinders, tori), it does not work for 'spherical' ones (i.e. manifolds with essential spheres). This is because such manifolds have no non-contractible loops, contrarily to 'circular' ones. Extending the tearing procedure to spherical manifolds is a challenging task and a research direction for future work.

Acknowledgements

The authors are grateful to the anonymous reviewers for their helpful comments and constructive suggestions.

References

[1] A. Ahalt, A.K. Krishnamurthy, P. Chen, D.E. Melton, Competitive learning algorithms for vector quantization, Neural Networks 3 (1990) 277–290.
[2] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.
[3] M. Bernstein, V. de Silva, J.C. Langford, J.B. Tenenbaum, Graph approximations to geodesics on embedded manifolds, Technical Report, Stanford University, Stanford, December 2000.
[4] P. Demartines, J. Hérault, Curvilinear Component Analysis: A self-organizing neural network for nonlinear mapping of data sets, IEEE Trans. Neural Networks 8 (1) (1997) 148–154.
[5] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Algorithms for drawing graphs: an annotated bibliography, Technical Report, Brown University, June 1994.
[6] E.W. Dijkstra, A note on two problems in connection with graphs, Numer. Math. 1 (1959) 269–271.
[7] D.L. Donoho, C. Grimes, Hessian eigenmaps: new locally linear techniques for high-dimensional data, Technical Report TR03-08, Department of Statistics, Stanford University, 2003.
[8] J. Hérault, C. Jaussions-Picaud, A. Guérin-Dugué Curvilinear component analysis for high dimensional data representation: I, Theoretical aspects and practical use in the presence of noise, in: J. Mira, J.V. Sánchez, (Eds.), Proceedings of IWANN'99, vol. II, Springer, Alicante, Spain, June 1999, pp. 635–644.
[9] J.B. Kruskal, On the shortest spanning tree of a graph and the traveling salesman problem, Proc. Am. Math. Soc. 7 (1956) 48–50.
[10] J.A. Lee, C. Archambeau, M. Verleysen, Locally linear embedding versus isotop, in: M. Verleysen (Ed.), Proceedings of ESANN'2003, 11th European Symposium on Artificial Neural Networks, D-Side public., Bruges, Belgium, April 2003, pp. 527–534.
[11] J.A. Lee, A. Lendasse, M. Verleysen, Curvilinear distances analysis versus isomap, in: M. Verleysen (Ed.), Proceedings of ESANN'2002, 10th European Symposium on Artificial Neural Networks, D-Facto public., Bruges, Belgium, 2002, pp. 13–20.
[12] J.A. Lee, M. Verleysen, Nonlinear projection with the isotop method, in: J.R. Dorronsoro (Ed.), Artificial Neural Networks, Lecture Notes in Computer Science, vol. 2415, Proceedings of ICANN 2002, Springer, Madrid, Spain, August 2002, pp. 933–938.
[13] J.A. Lee, M. Verleysen, Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis, Neurocomputing 57 (2004) 49–76.
[14] R.C. Prim, Shortest connection networks and some generalisations, Bell System Tech. J. 36 (1957) 1389–1401.
[15] S.T. Roweis, L.K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (55) (2000) 2323–2326.
[16] J.W. Sammon, A nonlinear mapping algorithm for data structure analysis, IEEE Trans. Comput. CC-18 (5) (1969) 401–409.
[17] L.K. Saul, S.T. Roweis, Think globally, fit locally: unsupervised learning of nonlinear manifolds, Technical Report, University of Pennsylviana, Philadelphia, 2002.
[18] L.K. Saul, S.T. Roweis, Think globally, fit locally: unsupervised learning of low-dimensional manifolds, J. Mach. Learning Res. 4 (2003) 119–155.
[19] M. Stoer, F. Wagner, A simple min-cut algorithm, J. ACM 44 (4) (1997) 585–591.
[20] J.B. Tenenbaum, Mapping a manifold of perceptual observations, in: M. Jordan, M. Kearns, S. Solla (Eds.), Advances in Neural Information Processing Systems, vol. 10, MIT Press, Cambridge, MA, 1998, pp. 682–688.

[21] J.B. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.

John A. Lee was born in 1976 in Brussels, Belgium. He received the M.Sc. degree in Applied Sciences (Computer Engineering) in 1999 and the Ph.D. degree in Applied Sciences (Machine Learning) in 2003, both from the Université catholique de Louvain (Belgium). His main interests are nonlinear dimensionality reduction, intrinsic dimensionality estimation, independent component analysis, cluster analysis and vector quantization. He is now working as a post-doc researcher on medical image segmentation by means of clustering methods in the Saint-Luc University Hospital (Belgium).

Michel Verleysen was born in 1965 in Belgium. He received the M.S. and Ph.D. degrees in Electrical Engineering from the Université catholique de Louvain (Belgium) in 1987 and 1992, respectively. He was an Invited Professor at the Swiss E.P.F.L. (Ecole Polytechnique Fédérale de Lausanne, Switzerland) in 1992, at the Université d'Evry Val d'Essonne (France) in 2001, and at the Université ParisI-Panthéon-Sorbonne in 2002, 2003 and 2004. He is now a Senior Research Associate of the Belgian F.N.R.S. (Fonds National de la Recherche Scientique) and Lecturer at the Université catholique de Louvain. He is editor-in-chief of the Neural Processing Letters journal and chairman of the annual ESANN conference (European Symposium on Artificial Neural Networks); he is associate editor of the IEEE Transactions on Neural Networks Journal, and member of the editorial board and program committee of several journals and conferences on neural networks and learning. He is author or co-author of about 160 scientific papers in international journals and books or communications to conferences with reviewing committee. He is the co-author of the scientific popularization book on artificial neural networks in the series "Que Sais-Je?", in French. His research interests artificial neural networks, self-organization, time-series forecasting, nonlinear statistics, adaptive signal processing, and electronic implementations of neural and biomedical systems.