compound of two functions. The first one depends on the distance in the embedding space, as for regular CCA, whereas the second has for argument the distance in the data space, as for Sammon's NLM:

$$F_\lambda(d_\mathbf{x}, d_\mathbf{y}) = (1-\rho)H(\lambda - d_\mathbf{x}) + \rho H(\lambda - d_\mathbf{y}) \ . \tag{4.93}$$

The additional parameter ρ allows one to tune the balance between both terms. With ρ close to one, local neighborhoods are generally well preserved, but discontinuities can possibly appear (manifold tears). In contrast, with ρ close to zero, the global manifold shape is better preserved, often at the price of some errors in local neighborhoods.

Finally, a method similar to CCA is described in [59]. Actually, this method is more closely related to VQP [46, 45] than to CCA as it is described in [48]. This method relies on the neural gas [135] for the vector quantization and is thus able to work online (i.e., with a time-varying data set).

4.3 Graph distances

Briefly put, graph distances attempt to overcome some shortcomings of spatial metrics like the Euclidean distance. The next subsection introduces both the geodesic and graph distances, explains how they relate to each other, and motivates their use in the context of dimensionality reduction. The subsequent subsections describe three methods of nonlinear dimensionality reduction that use graph distances.

4.3.1 Geodesic distance and graph distance

In order to reduce the dimensionality of highly folded manifolds, algorithms like NLM and CCA extend the purely linear MDS mainly by changing the optimization procedure. Instead of an exact solution computed algebraically, these algorithms use more sophisticated optimization procedures. This gives more freedom in the definition of the error function, for example, by allowing the user to differently weight short and long distances.

Fundamentally, however, the problem of unfolding may be solved from the opposite side. The goal of NLM and CCA is formalized by complex error functions that preserve short distance and allow the stretching of longer ones. But is this not an awkward remedy to the fact that traditional spatial metrics like the Euclidean one are not adapted to distance preservation? Would it be possible, just by changing the metric used to measure the pairwise distances, either to keep the simplicity of metric MDS or to attain better performances? These two directions are explored respectively by Isomap (Subsection 4.3.2) and curvilinear distance analysis (Subsection 4.3.4).

The idea of these two methods results from simple geometrical considerations, as illustrated in Fig. 4.9. That figure shows a curved line, i.e., a

one-dimensional manifold embedded in a two-dimensional space. In order to reduce the dimensionality to 1, it is intuitively expected that the curve has to be unrolled. Assuming that very short Euclidean distances are preserved, this means, as a counterpart, that Euclidean longer distances are considerably stretched. For example, the distance between the two endpoints of the C curve in Fig. 4.9 would be multiplied by more than three! Intuitively, such an issue could be addressed by measuring the distance *along the manifold* and not *through the embedding space*, like the Euclidean distance does. With such a metric, the distance depends less on the particular embedding of the manifold. In simpler words, the curvature of the manifold does not modify (or hardly modifies) the value of the distance. The distance along a manifold is usually called the *geodesic distance*, by analogy with curves drawn on the surface of Earth. The geodesic distance can also be interpreted as a railroad distance: trains are forced to follow the track (the manifold). On the other hand, the Euclidean distances may follow straight shortcuts, as a plane flies regardless of roads and tracks.

Formally, the geodesic distance is rather complicated to compute starting with the analytical expression of a manifold. For example, in the case of a one-dimensional manifold $\mathcal{M}$, which depends on a single latent variable x, like the C curve, parametric equations can be written as

$$\mathbb{R} \to \mathcal{M} \subset \mathbb{R}^D : x \mapsto \mathbf{m}(x) = [m_1(x), \ldots, m_D(x)]^T \tag{4.94}$$

and may help to compute the manifold distance as an arc length. Indeed, the arc length l from point $\mathbf{y}(i) = \mathbf{m}(x(i))$ to point $\mathbf{y}(j) = \mathbf{m}(x(j))$ is computed as the integral

$$l = \int_{\mathbf{y}(i)}^{\mathbf{y}(j)} dl = \int_{\mathbf{y}(i)}^{\mathbf{y}(j)} \sqrt{\sum_{k=1}^{D} dm_i^2} = \int_{x(i)}^{x(j)} \|\mathbf{J}_x \mathbf{m}(x)\| dx \ , \tag{4.95}$$

where $\mathbf{J}_x \mathbf{m}(x)$ designates the Jacobian matrix of $\mathbf{m}$ with respect to the parameter x. As x is scalar, the Jacobian matrix reduces to a column vector.

Unfortunately, the situation gets worse for multidimensional manifolds, which involve more than one parameter:

$$\mathbf{m} : \mathbb{R}^P \to \mathcal{M} \subset \mathbb{R}^D, \mathbf{x} = [x_1, \ldots, x_P]^T \mapsto \mathbf{y} = \mathbf{m}(\mathbf{x}) = [m_1(\mathbf{x}), \ldots, m_D(\mathbf{x})]^T \ . \tag{4.96}$$

In this case, several different paths may go from point $\mathbf{y}(i)$ to point $\mathbf{y}(j)$. Each of these paths is in fact a one-dimensional submanifold $\mathcal{P}$ of $\mathcal{M}$ with parametric equations

$$\mathbf{p} : \mathbb{R} \to \mathcal{P} \subset \mathcal{M} \subset \mathbb{R}^P, \quad z \mapsto \mathbf{x} = \mathbf{p}(z) = [p_1(z), \ldots, p_P(z)]^T \ . \tag{4.97}$$

The integral then has to be minimized over all possible paths that connect the starting and ending points:

$$l = \min_{\mathbf{p}(z)} \int_{z(i)}^{z(j)} \|\mathbf{J}_z \mathbf{m}(\mathbf{p}(z))\| dz \ . \tag{4.98}$$

In practice, such a minimization is intractable since it is a functional minimization. Anyway, the parametric equations of $\mathcal{M}$ (and $\mathcal{P}$) are unknown; only some (noisy) points of $\mathcal{M}$ are available.

Although this lack of analytical information is usually considered as a drawback, this situation simplifies the problem of the arc length minimization because it becomes discrete. Consequently, the problem has to be reformulated. Instead of minimizing an arc length between two points on a manifold, the task becomes the following: minimize the length of a path (i.e., a broken line) from point $\mathbf{y}(i)$ to point $\mathbf{y}(j)$ and passing through a certain number of other points $\mathbf{y}(k), \mathbf{y}(l), \ldots$ of the manifold $\mathcal{M}$. Obviously, the Euclidean distance is a trivial solution to this problem: the path directly connects $\mathbf{y}(i)$ to $\mathbf{y}(j)$. However, the idea is that the path should be constrained to follow the underlying manifold, at least approximately. Therefore, a path cannot jump from one point of the manifold to any other one. Restrictions have to be set. Intuitively, they are quite simple to establish. In order to obtain a good approximation of the true arc length, a fine discretization of the manifold is needed. Thus, only the smallest jumps will be permitted. In practice, several simple rules can achieve this goal. A first example is the K-rule, which allows jumping from one point to the K closest other ones, K being some constant. A second example is the ϵ-rule, which allows jumping from one point to all other ones lying inside a ball with a predetermined radius ϵ. See Appendix E for more details about these rules.

Formally, the set of data points associated with the set of allowed jumps constitutes a graph in the mathematical sense of the term. More precisely, a *graph* is a pair $G = (V_N, E)$, where V_N is a finite set of N *vertices* v_i (intuitively: the data points) and E is a set of *edges* (intuitively: the allowed jumps). The edges are encoded as pairs of vertices. If edges are ordered pairs, then the graph is said to be *directed*; otherwise it is *undirected*. In the present case, the graph is *vertex-labeled*, i.e., there is a one-to-one correspondence between the N vertices and the N data points $\mathbf{y}(i)$: $\text{label}(v_i) = \mathbf{y}(i)$. In order to compute the length of paths, the graph has to be vertex-labeled as well as *edge-labeled*, meaning in the present case that edges are given a length. As the length is a numerical attribute, the graph is said to be (edge-)weighted. If the edge length is its Euclidean length, then the graph is said to be Euclidean and

$$\text{label}\left((v_i, v_j)\right) = \|\text{label}(v_i) - \text{label}(v_j)\|_2 = \|\mathbf{y}(i) - \mathbf{y}(j)\|_2 \ . \tag{4.99}$$

A path π in a graph G is an ordered subset of vertices $[v_i, v_j, v_k, \ldots]$ such that the edges $(v_i, v_j), (v_j, v_k), \ldots$ belong to E. In a Euclidean graph, it is sometimes easier to write the same path π by the short-hand notation $\pi = [\mathbf{y}(i), \mathbf{y}(j), \mathbf{y}(k), \ldots]$. The length of π is defined as the sum of the lengths of its constituting edges:

$$\text{length}(\pi) = \text{label}((v_i, v_j)) + \text{label}((v_j, v_k)) + \ldots \quad (4.100)$$

In order to be considered as a distance, the path length must show the properties of nonnegativity, symmetry, and triangular inequality (see Subsection 4.2.1). Non-negativity is trivially ensured since the path length is a sum of Euclidean distances that already show this property. Symmetry is gained when the graph is undirected. In the case of the K-rule, this means in practice that if point $\mathbf{y}(j)$ belongs to the K closest ones from point $\mathbf{y}(i)$, then the edge from $\mathbf{y}(i)$ to $\mathbf{y}(j)$ as well as the edge from $\mathbf{y}(j)$ to $\mathbf{y}(i)$ are added to E, even if point $\mathbf{y}(i)$ does not belong to the K closest ones from $\mathbf{y}(j)$. The triangular inequality requires that, given points $\mathbf{y}(i)$ and $\mathbf{y}(j)$, the length computed between them must follow the shortest path. If this condition is satisfied, then

$$\text{length}([\mathbf{y}(i), \ldots, \mathbf{y}(j)]) \leq \text{length}([\mathbf{y}(i), \ldots, \mathbf{y}(k)]) + \text{length}([\mathbf{y}(k), \ldots, \mathbf{y}(j)]) \quad (4.101)$$

holds. Otherwise, the concatenated path $[\mathbf{y}(i) \ldots, \mathbf{y}(k), \ldots, \mathbf{y}(j)]$ would be shorter than $\text{length}([\mathbf{y}(i) \ldots, \mathbf{y}(j)])$, what is impossible by construction.

At this point, the only remaining problem is how to compute the shortest paths in a weighted graph? Dijkstra [53] designed an algorithm for this purpose. Actually, Dijkstra's procedure solves the *single-source shortest path* problem (SSSP). In other words, it computes the shortest paths between one predetermined point (the source) and all other ones. The *all-pairs shortest path* problem (APSP) is easily solved by repeatedly running Dijkstra's procedure for each possible source [210]. The only requirement of the procedure if the nonnegativity of the edge labels. The result of the procedure is the length of the shortest paths from the source to all other vertices. A representation of the shortest paths can be computed as a direct byproduct. In graph theory, the lengths of the shortest paths are traditionally called *graph distances* and noted

$$\delta(\mathbf{y}(i), \mathbf{y}(j)) = \delta(v_i, v_j) = \min_{\pi} \text{length}(\pi) \ , \quad (4.102)$$

where $\pi = [v_i, \ldots, v_j]$.

Dijkstra's algorithm works as follows. The set of vertices V is divided into two subsets such that $V_N = V_{\text{done}} \cup V_{\text{sort}}$ and $\emptyset = V_{\text{done}} \cap V_{\text{sort}}$ where

- V_{done} contains the vertices for which the shortest path is known;
- V_{sort} contains the vertices for which the shortest path is either not known at all or not completely known.

If v_i is the source vertex, then the initial state of the algorithm is $V_{\text{done}} = \emptyset$, $V_{\text{sort}} = V_N$, $\delta(v_i, v_i) = 0$, and $\delta(v_i, v_j) = \infty$ for all $j \neq i$. After the initialization, the algorithm iteratively looks for the vertex v_j with the shortest $\delta(v_i, v_j)$ in V_{sort}, removes it from V_{sort}, and puts it in V_{done}. The distance $\delta(v_i, v_j)$ is then definitely known. Next, all vertices v_k connected to v_j, i.e., such that $(v_j, v_k) \in E$, are analyzed: if $\delta(v_i, v_j) + \text{label}((v_j, v_k)) < \delta(v_i, v_k)$, then the value of $\delta(v_i, v_k)$ is changed to $\delta(v_i, v_j) + \text{label}((v_j, v_k))$ and the candidate shortest path from v_i to v_k becomes $[v_i, \ldots, v_j, v_k]$, where $[v_i, \ldots, v_j]$ is the

shortest path from v_i to v_j. The algorithm stops when $V_{\text{sort}} = \emptyset$. If the graph is not connected, i.e., if there are one or several pairs of vertices that cannot be connected by any path, then some $\delta(v_i, v_j)$ keep an infinite value.

Intuitively, the correctness of the algorithm is demonstrated by proving that vertex v_j in V_{sort} with shortest path $\pi_1 = [v_i, \ldots, v_j]$ of length $\delta(v_i, v_j)$ can be admitted in V_{done}. This is trivially true for v_i just after the initialization. In the general case, it may be assumed that an unexplored path going from the source v_i to vertex v_j is shorter than the path found by Dijkstra's algorithm. Then this hypothetical "shorter-than-shortest" path may be written as $\pi_2 = [v_i, \ldots, v_t, v_u, \ldots, v_j]$, wherein $\pi_3 = [v_i, \ldots, v_t]$ is the maximal (always non-empty) subpath belonging to V_{done} and $\pi_4 = [v_u, \ldots, v_j]$ is the unexplored part of π_2. Consequently, v_u still lies in V_{sort} and $\delta(v_i, v_u)$ has been given its value when vertex v_t was removed from V_{sort}. Thus, the inequalities

$$\text{length}(\pi_3) < \text{length}(\pi_2) < \text{length}(\pi_1) \tag{4.103}$$

hold but contradict the fact that the first path π_1 was chosen as the one with the shortest $\delta(v_i, v_j)$ in V_{sort}.

At this point, it remains to prove that the graph distance approximates the true geodesic distance in an appropriate way. Visually, it seems to be true for the C curve, as illustrated in Fig. 4.10. Formal demonstrations are provided in [19], but they are rather complicated and not reproduced here. The intuitive idea consists of relating the natural Riemannian structure of a smooth manifold $\mathcal{M}$ to the graph distance computed between points of $\mathcal{M}$. Bounds are more easily computed for graphs constructed with ϵ-balls than with K-ary neighborhoods. Unfortunately, these bounds rely on assumptions that are hard to meet with real data sets, especially for K-ary neighborhoods. Moreover, the bounds are computed in the ideal case where no noise pollutes the data.