Tangent Space Guided Intelligent Neighbor Finding

Mike Gashler and Tony Martinez

Abstract— **We present an intelligent neighbor-finding algorithm called SAFFRON that chooses neighboring points while avoiding making connections between points on geodesically distant regions of a manifold. SAFFRON identifies the suitability of points to be neighbors by using a relaxation technique that alternately estimates the tangent space at each point, and measures how well the estimated tangent spaces align with each other. This technique enables SAFFRON to form high-quality local neighborhoods, even on manifolds that pass very close to themselves. SAFFRON is even able to find neighborhoods that correctly follow the manifold topology of certain self-intersecting manifolds.**

I. Introduction

Many algorithms commonly used in machine learning rely on local neighborhoods of points. Instance-based learners (IBL), for example, use consensus among neighboring points to determine a predicted label for previously unseen points. As another example, non-linear dimensionality reduction (NLDR) algorithms measure the distances between neighboring points to represent structure on the surface of a manifold. The quality of results obtained by these algorithms is limited by the quality of the local neighborhoods with which they operate. When data is known to lie on the surface of a low-dimensional manifold in high-dimensional space, poor neighborhood connections are typically defined as those that shortcut across the manifold structure between geodesically distant regions of the manifold [1], [2].

We present a novel algorithm called Similarly Aligned Friend Finding RelaxatiON (*SAFFRON*) that is designed to find neighbors on the surface of a non-linear manifold. SAFFRON takes advantage of the assumption that the samples are drawn from the surface of a lower-dimensional manifold to select neighbors in a manner that intelligently avoids short-cutting to geodesically distant regions of the manifold surface. This is useful because many interesting data sets, including collections of images or documents, tend to form a non-linear manifold, and SAFFRON enables good neighbors to be found in such data, even when the manifold structure passes very near to, or in some cases even intersects with, itself. SAFFRON makes IBL and NLDR algorithms suitable for use with a wider class of problems–specifically, those that sample highly folded manifolds. In cases where the manifold structure does not fold back on itself, SAFFRON gives results similar to those obtained with Euclidean distance. Thus, it is well-suited for general-purpose use.

As a simple example of data the might lie on a manifold that passes very near to itself, we consider a hypothetical collection of images that is obtained by a robot with a camera as it travels down a hallway inside a building. It might be useful to use an NLDR algorithm to reduce this collection of images into fewer dimensions because the intrinsic variables in this data would correspond with the robot's position and orientation. Unfortunately, in many buildings, hallways contain doors that are very similar in appearance to each other. Thus, if Euclidean distance is used to find neighboring images, it might incorrectly determine that two images are very similar, even though they depict very distant positions with the hallway. These images may be considered to be samples from a highly-folded manifold with a topology that passes very near to itself. Existing techniques may not be able to find local neighborhoods on such a topology that correctly represent the structure of the manifold.

The SAFFRON algorithm seeks to intelligently select local neighborhoods that correctly represent the manifold structure, particularly when the manifold passes very close to itself or even intersects itself. We assume that a set of points, $\mathbf{P}$, has been sampled on the surface of a manifold, such that each point $\mathbf{p}_i \in \mathbf{P}$ is a vector in $\Re^d$. We also assume that the tangent-space dimensionality, t, of the manifold is known. (In cases where t is not known, methods exist for estimating this value from the data [3].) We refer to the points that SAFFRON determines to be compatible as *friends*, rather than *neighbors*, because they are determined by their *compatability*, rather than just by their proximity. SAFFRON determines two points to be compatible if the tangent spaces associated with them are closely aligned.

In order to accurately measure the tangent space at a point, the structure of the manifold would need to be known *a priori*. Unfortunately, most existing techniques for learning a manifold structure rely on first finding neighborhoods to represent the local structure. Thus, in order to identify points that are well-suited to be labeled as friends, SAFFRON uses a relaxation technique that alternately estimates the tangent space for each point from its current set of friends, and then refines the friends based on the alignment of their tangent spaces. When convergence is detected, SAFFRON can intelligently identify local neighborhoods in a manner that is unlikely to shorcut across the manifold topology. A high-level flow diagram of the SAFFRON algorithm is given in Figure 1.

The SAFFRON algorithm is comprised of six high-level steps. We now briefly discuss each of these high-level steps. A more detailed specification of the algorithm, including all details necessary for implementation, is given in Section III. The first high-level step of the SAFFRON algorithm uses Euclidean distance to find a set of candidate friends, $\mathbf{C}_i$. $|\mathbf{C}_i|$ should be larger than the number of friends, k, that SAFFRON seeks. A weight vector, $\mathbf{w}_i$ specifies the affinity

Mike Gashler and Tony Martinez are with the Department of Computer Science, Brigham Young University, Provo, Utah, U.S.A. (email: {mike, martinez}@axon.cs.byu.edu).

978-1-4244-9636-5/11/$26.00 ©2011 IEEE

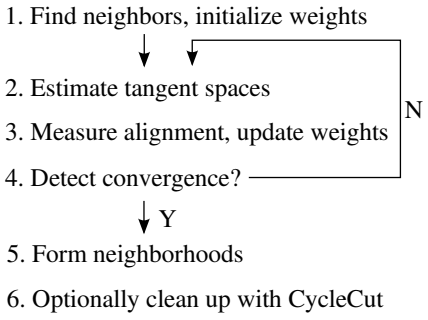


Fig. 1. A high-level flow diagram of the SAFFRON algorithm.

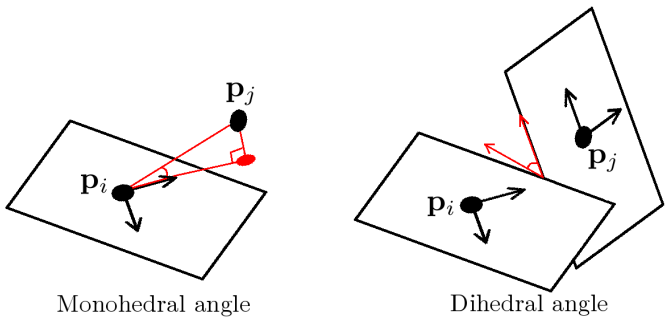


Fig. 2. The monohedral (one surface) angle is defined between a point with its corresponding tangent space, and another point. It is independent of the second point's tangent space, and is not commutative. The dihedral (two surface) angle is defined between two tangent spaces. It is independent of the points, and is commutative.

between $\mathbf{p}_i$ and each of its candidate friends in $\mathbf{C}_i$. Initially, each candidate point is given uniform weight. As the system relaxes in subsequent steps, the weight will tend to shift toward the k points in $\mathbf{C}_i$ whose tangent spaces are most aligned with that of $\mathbf{p}_i$.

The second high-level step of SAFFRON estimates the tangent space at each $\mathbf{p}_i \in \mathbf{P}$. This is done by computing the t-first principal components of $\mathbf{C}_i$. The weight of each neighbor is used when computing the principal components, such that as a candidate point gains more weight, it will have more influence on the estimate of the tangent space.

The third step is the most significant part of SAFFRON, so we describe it in greater detail. This step measures how well the tangent-space of $\mathbf{p}_i$ is aligned with the tangent-space of each candidate point, and uses this information to update the weights. In order to establish a suitable metric for determining how well the tangent spaces of two points are aligned, we define two types of angles as illustrated in Figure 2.

The monohedral (one-surface) angle is defined between a point, $\mathbf{p}_i$, with its corresponding tangent space, $\mathbf{S}_i$, and another point, $\mathbf{p}_j$. $\mathbf{S}_i$ is represented such that each row specifies one of the orthonormal basis vectors in the tangent space of $\mathbf{p}_i$. The monohedral angle is computed by projecting point $\mathbf{p}_j$ onto $\mathbf{S}_i$, and then computing the angle formed by the three points $\langle \mathbf{p}_j, \mathbf{p}_i, \mathbf{p}_i + \mathbf{S}_i(\mathbf{p}_j - \mathbf{p}_i) \rangle$. Equation 1 computes the cosine of the monohedral angle.

$$m(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j) = \frac{||\mathbf{S}_i(\mathbf{p}_j - \mathbf{p}_i)||}{||\mathbf{p}_j - \mathbf{p}_i||} \tag{1}$$

The dihedral (two-surfaces) angle is defined between two tangent spaces, $\mathbf{S}_i$ and $\mathbf{S}_j$. It is independent of the points $\mathbf{p}_i$ and $\mathbf{p}_j$. In the case where $t = d - 1$, the dihedral angle is simply the angle formed by the normal vectors of the two surfaces. Typically, however $t < d - 1$. In this case, we use the normal vectors that maximize their distance with their projection onto the other surface. This can be found with an eigenvector optimization technique. Equation 2 computes the cosine of the dihedral angle between two tangent spaces. This equation works even when the tangent spaces have a codimensionality greater than 1. The function $\mathrm{fpc}(\mathbf{M})$ as used in Equation 2 returns the first principal component of the row vectors in $\mathbf{M}$, or the eigenvector with the largest eigenvalue of the covariance matrix of $\mathbf{M}$.

$$\begin{aligned} d(\mathbf{S}_i, \mathbf{S}_j) = &\left(\mathbf{S}_i \,\mathrm{fpc}(\mathbf{S}_j^T \mathbf{S}_j \mathbf{S}_i^T - \mathbf{S}_i^T)\right) \cdot \\ &\left(\mathbf{S}_j \,\mathrm{fpc}(\mathbf{S}_i^T \mathbf{S}_i \mathbf{S}_j^T - \mathbf{S}_j^T)\right) \end{aligned} \tag{2}$$

SAFFRON uses Equation 3 to measure how well two tangent spaces are aligned. When $\mathbf{S}_i$ and $\mathbf{S}_j$ are mis-aligned, Equation 3 will return a value close to 0. When they are aligned, it will return a larger value. The monohedral angle is measured in both directions because it is not commutative, while the dihedral angle need only be evaluated once because it is commutative. The dihedral component of this equation will be small when the two tangent spaces are not approximately parallel. At least one of the two monohedral components will be small when the tangent spaces are approximately parallel, but have a large gap between them. Thus, when the product of all three components is large, the tangent spaces are necessarily aligned, and so Equation 3 provides a useful indication of whether the Euclidean distance between two points is representative of the geodesic distance between them, given the estimated tangent spaces, or whether those points actually lie on separate regions of a manifold that happens to fold back on itself to create a misleading proximity between the two points.

$$\begin{aligned} a(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j, \mathbf{S}_j, \lambda) = &\min(\lambda, m(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j)^2) * \\ &\min(\lambda, m(\mathbf{p}_j, \mathbf{S}_j, \mathbf{p}_i)^2) * \\ &\min(\lambda, d(\mathbf{S}_i, \mathbf{S}_j)^2) \end{aligned} \tag{3}$$

Equation 3 uses the squared cosine of these angles angles instead of just the cosine of these angles because both positive and negative correlations are equally indicative of alignment. In order to be tolerant of noise and curvature in the manifold, we cap the value of all three components with a threshold, λ, such that all nearly-aligned tangent spaces are evaluated as being equally good. λ can range from 0 to 1, but for most problems, suitable values typically range from about 0.85 to 0.95. Smaller values will cause the metric to be more

tolerant of curvature in the manifold, while larger values will cause the metric to be more careful to avoid connecting two points from geodesically distant regions of the manifold. As λ approaches 0, the results from SAFFRON approach those obtained using Euclidean distance to find neighbors.

Figure 3 gives an intuition for why the product of these three components provides a good measure of tangent-space alignment. This figure shows 4 cases. In each case, the two points are nearly the same distance apart, but their corresponding tangent spaces are oriented in various ways. For the purpose of visualization, we enclose each point with a rectangular region on its corresponding tangent space. Arrows extending from the point represent a set of basis vectors that might define this tangent space. In case 1, where the tangent spaces are aligned, all three values are close to 1. In each case where the the tangent spaces are misaligned in some way (cases 2, 3, and 4), at least one of the three values is close to zero. Thus, the product of these three components is only close to 1 when the tangent spaces are aligned.

SAFFRON decays the weights of the $|\mathbf{C}_i| - k$ candidate neighbors that are the least aligned with $\mathbf{p}_i$. The weight vector is then normalized to sum to 1. (This effectively shifts the weight toward the k candidate neighbors with tangent spaces that are most aligned with $\mathbf{p}_i$.)

Step 4 of the SAFFRON algorithm detects convergence. Convergence can be easily detected because each iteration increases the alignment scores of the candidate neighbors with the largest weights. Thus, a goodness score for the system is given as the sum of the alignment scores scaled by the weights. When this goodness value no longer increase significantly, it has converged.

Step 5 forms local neighborhoods of points. This is done for each point by selecting the k candidate neighbors with the largest weights.

The last high-level step of SAFFRON is to post-process the neighborhoods with the *CycleCut* algorithm [4]. CycleCut is an algorithm that uses a max-flow/min-cut technique to detect and prune neighbor connections that shortcut across the manifold topology. We consider a complete description of the CycleCut algorithm to be outside the scope of this paper, but we give a high-level overview of its function here. Also, for completeness in describing the SAFFRON algorithm, pseudocode for the CycleCut algorithm is included in Appendix A.

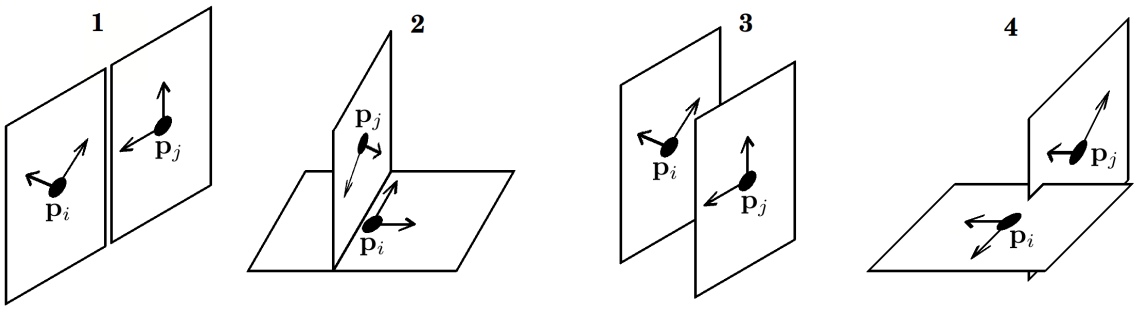
This step may be considered to be optional because with many problems, SAFFRON can form high-quality neighborhoods without this last step. The addition of this step, however, increases the robustness of the SAFFRON algorithm. SAFFRON relies on several parameter values (including the number of candidate points q, the number of neighbors k, the number of tangent-space dimensions t, and a threshold value, λ). When these values are poorly-tuned for the problem, or when the data contains a large amount of noise, SAFFRON may find a small number of neighbors that still shortcut across the manifold structure. CycleCut can detect and remove these connections, but it can only distinguish the shortcuts from valid connections if the number of shortcut connections is small. Thus, SAFFRON and CycleCut are designed to complement each other. SAFFRON makes it possible for CycleCut to identify shortcut connections by keeping the number of shortcut connections small, and CycleCut ensures that the final results will be free of the large cycles that are created by connections that shortcut across manifold boundaries.

The remainder of this paper is layed out as follows. Section II describes related work. Section III describes the SAFFRON and CycleCut algorithms in detail sufficient to facilitate implementation. Section IV reports results from empirical tests to validate the properties of SAFFRON. Finally, Section V summarizes the contributions of this paper.

II. Related Work

Due to the large number of algorithms that rely on being able to identify neighboring points, neighbor-finding has been a topic of interest in machine learning for a long time. Many distance, similarity, and dis-similarity metrics have been proposed for this purpose [5], [6], [7], [8], [9]. It has been shown that as dimensionality becomes large, the distance from a point to its farthest neighbor approaches the distance to its nearest neighbor [10]. Thus, techniques that intelligently select neighbors, rather than merely relying on a distance metric, are particularly important for applications involving high-dimensional space. In this paper, we focus on finding neighbors among points that are known to lie on the surface of an intrinsically low-dimensional manifold in high-dimensional space.

The idea of using the assumption that data lies on a manifold to guide the evaluation of distances is not new [11]. Recently, however, as interest in manifold learning has increased in machine learning communities, techniques for estimating distances between points that sample from an unknown manifold have become of interest. One example is *Ranking on Manifolds* [12]. This technique evaluates distance in a manner that penalizes paths that cross gaps over unsampled regions of the manifold, and thereby evaluates distance in a manner that conforms more closely to the intrinsic manifold structure. Perhaps the most similar algorithm to SAFFRON is Adaptive Neighborhood Selection for Manifold Learning [13]. This technique builds on top of Ranking on Manifolds by adaptively choosing parameter values in local neighborhoods. SAFFRON is similar in that it also adaptively chooses settings in each neighborhood, but differs in how it determines proximity to the manifold. SAFFRON uses an approach similar to that used by the Local Tangent Space Alignment (LTSA) [14] for estimating a tangent space by computing the first t principal components in local neighborhoods. Unlike LTSA, however, it without the assumption that neighbors have already been found, and it uses the tangent spaces to guide the selection of neighbors (or friends). This approach enables SAFFRON to find better local neighborhoods on the surface of sampled manifolds than existing techniques.



	Case 1	Case 2	Case 3	Case 4
$m(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j)$	≈ 1	≈ 0	≈ 0	≈ 1
$m(\mathbf{p}_j, \mathbf{S}_j, \mathbf{p}_i)$	≈ 1	≈ 1	≈ 0	≈ 1
$d(\mathbf{S}_i, \mathbf{S}_j)$	≈ 1	≈ 0	≈ 1	≈ 0
$a(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_j, \mathbf{S}_j, \lambda)$	$\approx \lambda^3$	≈ 0	≈ 0	≈ 0

Fig. 3. Four cases are shown with two points separated by a constant distance and their corresponding tangent spaces. Case 1 is the only case where the tangent spaces of the two points are aligned. The cosine of the monohedral angle is evaluated in both directions because it is not commutative, whereas the cosine of the dihedral angle is the same in both directions. The tangent spaces of two points are aligned when all three metrics are close to 1. This property generalizes into arbitrary dimensional space.

III. The SAFFRON algorithm

Figure 4 gives pseudocode for the SAFFRON algorithm. The parameters to the algorithm are:

$\mathbf{P} \equiv$ set of points on which to operate.
$q \equiv$ median number of candidate points from which to select friends.
$k \equiv$ number of friends to find for each point. ($k < q$.)
$t \equiv$ dimensionality of the manifold tangent space.
$\lambda \equiv$ threshold that determines tolerance to curvature.

Line 1 of the SAFFRON algorithm computes a radius, r, in which to find candidate friends for each point. Because a suitable radius is necessarily problem-specific, it is preferable to parameterize the algorithm in terms of q, the median number of candidates, and to compute r from q. **Lines 2-5** find a set of candidate friends for each point and assign uniform weight to each of them. These candidates consiste of all points within the specified radius of each point. **Lines 6, 10, and 20** compute a "goodness" score for the system that is used in line 21 to dectect convergence. The value of g will increase with each iteration until the system converges. **Lines 7-21** are the main loop of the SAFFRON algorithm. **Lines 8-9** estimate the tangent space of each point by computing the first t principal components of $\mathbf{C}_i$. Each candidate neighbors is weighted according to $\mathbf{w}_i$ when the principal components are computed. Each row in $\mathbf{S}_i$, therefore, will store one of the t orthonormal basis vectors in the tangent space for $\mathbf{p}_i$. **Lines 11-19** update the weights of each point in $\mathbf{P}$. **Line 15** uses Equation 3 to compute an alignment score for each candidate friend of $\mathbf{p}_i$. **Line 18** decays the weight of the candidate friends with the lowest alignment scores. Our implementation decays these weights by multiplying by 0.9. Obviously other values could be used here, but we have not been able to detect any significant effect on results by adjusting this value. If there are multiple candidates with the same alignment score, the closer points are considered to be better aligned. **Line 19** normalizes $\mathbf{w}_i$ to sum to one. Thus, lines 18 and 19 together effectively shift the weight toward the k-best friends of $\mathbf{p}_i$. **Line 21** detects convergence. Our implementation stops when the goodness score increases by less than 0.01% after one iteration over all of the data points. Other values could be used to detect convergence, but we have obtained little perceptible benefit by tuning this value. **Line 22** forms the local neighborhoods by selecting the k points with the largest weights to be the friends of each $\mathbf{p}_i$. **Line 23** performs a post-processing operation on the neighborhoods called *CycleCut*, which detects and removes any shortcut connections that were erroneously found in previous steps. CycleCut only is only effective at detecting shortcut connections if the number of shortcut connections is small, so it is typically not sufficient to just use CycleCut by itself. Pseudo-code for the CycleCut algorithm is given in Appendix A.

IV. Experimental Analysis

To show that SAFFRON is effective at forming neighborhoods that do not shortcut across the manifold, even when the manifold approaches very close to itself, we sampled 439 points uniformly on an intrinsically one-dimensional compressed helix manifold defined by the equations $\langle \sin(\alpha), 0.005\alpha, \cos(\alpha) \rangle$. Figure 5.A shows each point connected with its four nearest Euclidean-distance neighbors. If the manifold is considered to be an intrinsically two-dimensional ring, then the neighbors found by Euclidean distance correctly represent its topology. Unfortunately, Euclidean distance does not provide a mechanism to specify the intrinsic dimensionality. With SAFFRON, this is done by setting the parameter value t. Figure 5.B shows the same points connected with their 4-best friends, as determined by SAFFRON with parameters $q = 32$, $k = 4$, $t = 1$, $\lambda = 0.95$.

We note that a solution to a compressed helix manifold using adaptive neighborhood selection was also presented

	function $\mathbf{SAFFRON}(\mathbf{P}, q, k, t, \lambda)$	*Comments*
1	**set** $r \leftarrow$ the median distance to the q^{th} neighbor in $\mathbf{P}$.	*Compute a good radius to use.*
2	**for each** point $\mathbf{p}_i \in \mathbf{P}$:	
3	**let** $\mathbf{C}_i$ be the subset of points $\mathbf{p}_j \in \mathbf{P}$, where $j \neq i$, and $\|\|\mathbf{p}_j - \mathbf{p}_i\|\| < r$.	*Find a set of candidate neighbors for each* $\mathbf{p}_i$.
4	**let** $\mathbf{w}_i$ be a vector of weights for points in $\mathbf{C}_i$.	w_{ij} *is the weight of* $\mathbf{c}_{ij}$ *for* $\mathbf{p}_i$.
5	**set each** $w_{ij} \in \mathbf{w}_i \leftarrow \frac{1}{\|\mathbf{w}_i\|}$.	*Initialize each candidate to have uniform weight.*
6	$g \leftarrow 0$	g *is the increasing "goodness" of the system.*
7	**loop**:	
8	**for each** point $\mathbf{p}_i \in \mathbf{P}$:	
9	$\mathbf{S}_i \leftarrow$estimate_tangent_space$(\mathbf{P}, i, \mathbf{C}_i, \mathbf{w}_i, t)$.	*Estimate the weighted tangent space at each point.*
10	**set** $h \leftarrow g$; $g \leftarrow 0$.	*These values assist detecting convergence.*
11	**for each** point $\mathbf{p}_i \in \mathbf{P}$:	
12	**let** $\mathbf{x}_i$ be a vector of alignment scores for each candidate point in $\mathbf{C}_i$.	
13	**for each** point $\mathbf{c}_{ij} \in \mathbf{C}_i$:	
14	**let** $\mathbf{p}_l$ be the point in $\mathbf{P}$ that is $\mathbf{c}_{ij}$.	
15	**set** $x_{ij} \leftarrow a(\mathbf{p}_i, \mathbf{S}_i, \mathbf{p}_l, \mathbf{S}_l, \lambda)$.	*Equation 3.*
16	**set** $e \leftarrow \|\mathbf{x}\| - k$.	e *is the number of excess candidates.*
17	**for each** of the e-smallest values $x_{ij} \in \mathbf{x}_i$:	
18	**set** $w_{ij} \leftarrow 0.9 * w_{ij}$.	*Decay weight of least-aligned candidates.*
19	**set** $\mathbf{w}_i \leftarrow \frac{\mathbf{w}_i}{\sum_j w_{ij}}$.	*Normalize* $\mathbf{w}_i$ *to sum to* 1 *again.*
20	**set** $g \leftarrow g + \mathbf{x}_i \cdot \mathbf{w}_i$.	*Update the goodness measure.*
21	**until** $\frac{g}{h} - 1 < 0.0001$	*Terminate when* g *no longer increases significantly.*
22	**for each** $\mathbf{p}_i \in \mathbf{P}$, choose the k candidate points with the largest weights to be the friends of $\mathbf{p}_i$.	
23	Prune any shorcut connections with the CycleCut algorithm.	*See Appendix A.*

Fig. 4. Pseudo code for finding the k-best friends of each row in $\mathbf{P}$, assuming the points lie on a t-dimensional manifold. r is a radius in which to find candidate points. λ is a threshold value between 0 and 1 the specifies how well-aligned the tangent-spaces of points must be in order for them to be considered friends.

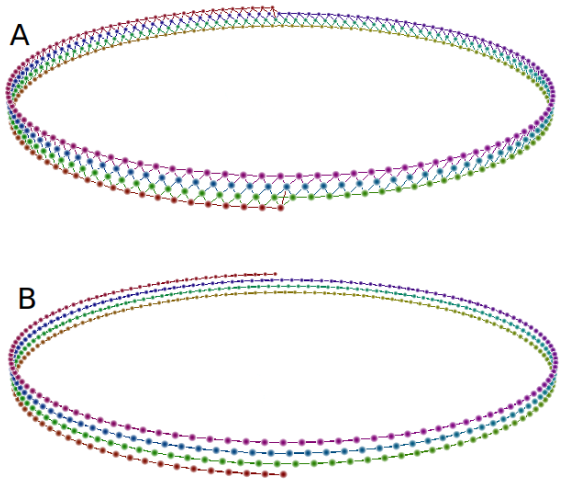


Fig. 5. **A:** Samples at regular intervals are shown on an intrinsically one-dimensional compressed helix manifold. Each point is shown connected with its four nearest Euclidean-distance neighbors. These would represent good local neighborhoods for an intrinsically two-dimensional ring manifold, but Euclidean distance does not provide a mechanism to specify the intrinsic dimensionality. **B:** Each point is shown connected with its four best friends as determined by SAFFRON with parameters $q = 32$, $k = 4$, $t = 1$, $\lambda = 0.95$.

by [13]. In that case, however, the compressed helix was compressed only to the point where the nearest neighbors of each point still included the previous and next points in the one-dimensional sequence. The compressed helix manifold for which we report results is a much harder problem because it is sufficiently compressed that neither the previous nor next point in the sequence is even among the four nearest Euclidean-distance neighbors for many of the points. By aligning tangent spaces, however, SAFFRON is able to correctly determine which points are most likely to be neighboring samples on a manifold with the specified number of intrinsic dimensions. If t is set to 2 instead of 1, SAFFRON behaves more like Euclidean distance by choosing neighbors in vertical directions as well as horizontal directions.

The parameters that we used to solve this problem were selected intuitively. The value $q = 32$ was chosen such that the pool of candidate friends for each point would be sufficiently large to include the two previous and two next points in the helix. The value $k = 4$ was chosen for the visual appeal of displaying 4 neighbors in Figure 5.A. The value $t = 1$ was chosen to indicate that the intended intrinsic dimensionality of the problem. The value $\lambda = 0.95$

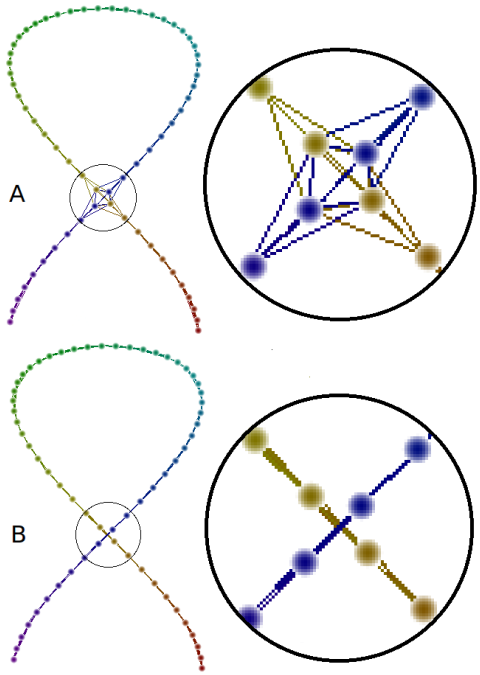


Fig. 6. **A:** Samples at regular intervals are shown on an intrinsically one-dimensional self-intersecting manifold. Each point is shown connected with its four nearest Euclidean-distance neighbors. These connections shortcut across the manifold structure, misrepresenting the manifold topology. NLDR algorithms that rely on such neighborhoods will not correctly unfold the manifold. **B:** Each point is shown connected with its four best friends as determined by SAFFRON with parameters $q = 6$, $k = 4$, $t = 1$, $\lambda = 0.9$.

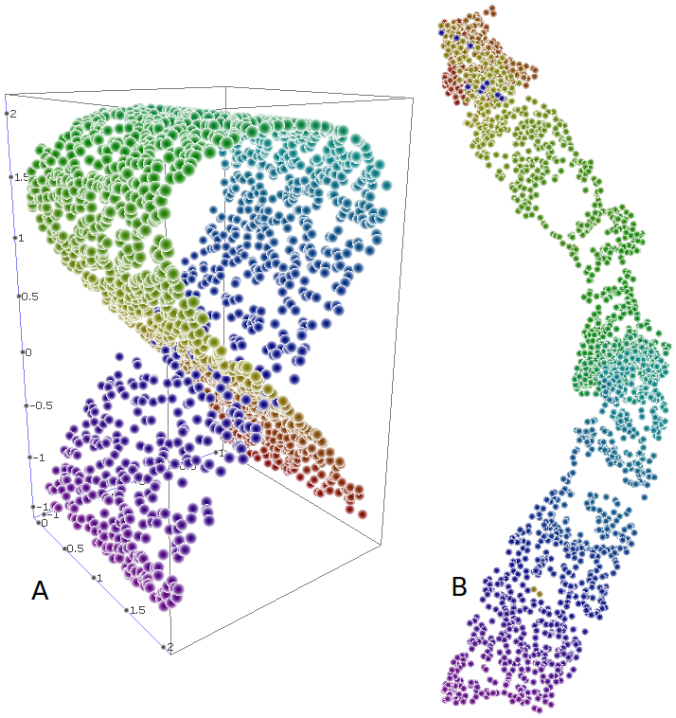


Fig. 7. **A:** Samples at regular intervals are shown on an intrinsically one-dimensional self-intersecting manifold. Each point is shown connected with its four nearest Euclidean-distance neighbors. These connections shortcut across the manifold structure, misrepresenting the manifold topology. NLDR algorithms that rely on such neighborhoods will not correctly unfold the manifold. **B:** Each point is shown connected with its four best friends as determined by SAFFRON with parameters $q = 6$, $k = 4$, $t = 1$, $\lambda = 0.9$.

was chosen to be fairly large in order to prevent neighbors from shortcutting across the manifold. With these parameters, SAFFRON made no shortcutting neighbor connections. The same results were obtained both with and without the final CycleCut shortcut-pruning step of the SAFFRON algorithm. When the CycleCut step is included, however, a broader range of parameter values can be used to still obtain the same results.

To test SAFFRON with an even more extreme manifold, we sampled 63 points from an intrinsically one-dimensional manifold that intersects with itself according to the equations $\langle \sin(2\alpha), 2\cos(\alpha) \rangle$. Figure 6.A shows each point connected with its 4-nearest Euclidean-distance neighbors. These connections shortcut across the manifold structure, misrepresenting the manifold topology. If an NLDR algorithm were to try to unfold this manifold based on these neighborhoods, the shortcut connections would cause it to produce incorrect results. By contrast, Figure 6.B shows the same points connected with their 4-best friends as determined by SAFFRON with parameters $q = 6$, $k = 4$, $t = 1$, $\lambda = 0.9$. None of the neighbor connections found by SAFFRON shortcut across the manifold topology. SAFFRON is able to determine that the very close points near the center would make poor friends because they have very mis-aligned tangent spaces. To our knowledge, SAFFRON is the first neighbor-finding technique that can correctly find neighbors on a self-intersecting manifold. The same results were obtained both with and without the final CycleCut shortcut-pruning step of the SAFFRON algorithm. When the CycleCut step is included, however, a broader range of parameter values can be used to still obtain the same results.

We also created an intrinsically two-dimensional manifold embedded in three-dimensional space according to the equations $\langle \sin(2\alpha), 2\cos(\alpha), 2\beta \rangle$. With this manifold, we sampled 2000 points using random values for α and β from a uniform distribution. Figure 7.A shows a plot of these point. We used SAFFRON to compute local neighborhoods with the parameters $q = 40$, $k = 18$, $t = 2$, $\lambda = 0.9$. We then used Manifold Sculpting [15] to reduce the dimensionality of these points to two dimensions, using the local neighborhoods computed by SAFFRON. Figure 7.B shows a plot of the data after reducing to two dimensions.

For visual clarity, the points in Figure 7 are colored according the the value of α. SAFFRON, of course, did not have access to any information that would enable it to determine the color of any of the points, or the corresponding intrinsic values α or β. It can be observed in Figure 7.B that a small number of purple-colored points were incorrectly placed among the yellow points, and a small number of yellow points were incorrectly placed among the purple point. This occurs because these points were located very near to where the manifold intersected itself. As the relaxation

```
function CycleCut(V, E)                                  Comments
    λ ← 12                                               Default atomic cycle length threshold
1.  for each {a, b} ∈ E do: W_{a,b} ← 1                  Initialize edge capacities uniformly
    R ← empty list                                       R stores the edges that are removed
    loop:                                                Loop until all large atomic cycles are cut
2.      C ← find_large_atomic_cycle(V, E, λ)             See Figure 9
        if C = null:                                     If there are no large atomic cycles
            break                                        Exit the loop. Go to *
3.      h ← min_{{a,b}∈C} W_{a,b}                        Find the bottle-neck in the cycle
4.      for each {a, b} ∈ C:                             For each edge in the cycle
            W_{a,b} ← W_{a,b} − h                        Reduce the remaining capacity
5.          if W_{a,b} = 0:                              If the edge is fully saturated
                remove {a, b} from E                     Cut the edge
                append {a, b} → R                        Remember the removed edges
6.      continue                                         Go to the start of the loop
7.  for each {a, b} ∈ R:                                 * Repair unnecessary cuts
        add {a, b} → E                                   Tentatively restore the edge
        C ← find_large_atomic_cycle(V, E, λ)             See Figure 9
        if C ≠ null:                                     If the edge creates a cycle
            remove {a, b} from E                         Remove it again
```

Fig. 8. Pseudo-code for the CycleCut algorithm. V is a set of vertices. E is a set of edges.

```
function find_large_atomic_cycle(V, E, λ)                Comments
Q ← empty queue; S ← empty set; T ← empty set            S = visited vertices, T = visited edges
choose a random vertex, v_0 from V                       Pick a random seed point
enqueue v_0 → Q; add v_0 → S                             Seed the outer BFS queue
while |Q| > 0:                                           Do the outer breadth-first-search
.   dequeue a ← Q                                        Visit the next vertex
.   for each neighbor b of a:                            Follow every edge
.       if b ∈ S:                                        If a cycle is detected by the outer BFS
.       .   P_b ← null                                   P = parent vertices
.       .   I ← empty queue; U ← empty set               U = vertices visited by inner BFS
.       .   enqueue b → I; add b → U                     Seed the inner BFS queue
.       .   while |I| > 0:                               Do the inner breadth-first-search
.       .       dequeue c ← I                            Visit the next vertex
.       .       for each neighbor d of c:                Follow every edge
.       .           if d ∉ U and E_{c,d} ∈ T:            Inner BFS does not explore new paths
.       .               P_d ← c                          Store the parent of every vertex
.       .               enqueue d → I; add d → U         Advance the inner BFS
.       .               if d = a:                        If an atomic cycle is found
.       .                   Y ← empty list               Y = list of vertices in the atomic cycle
.       .                   while d ≠ null:              Build the list of vertices
.       .                   .   append d → Y             Add to the list
.       .                   .   d ← P_d                  Advance to parent vertex
.       .                   if |Y| ≥ λ then return Y     Return the large atomic cycle
.       .                   else break twice             Exit inner BFS. Go to * (3 lines below)
.       else                                             If b has not been visited before
.       .   enqueue b → Q; add b → S                     Advance the outer BFS
.       add E_{a,b} → T                                  * Flag this edge as visited
return null                                              There are no large atomic cycles
```

Fig. 9. Pseudo-code to find an atomic cycle with a cycle length $\geq \lambda$ edges in a graph comprised of vertices V, and undirected edges E.

process used by SAFFRON proceeds, the tangent spaces at each point align themselves with their neighbors. With these points, the correct alignment could not be unambiguously determined. As these points began to align with one side, they also simultaneously disassociated themselves with the other side, and therefore became entirely embedded into just one location of the graph. A less-intelligent neighbor-finding technique might leave these points connected to both regions of the manifold. This would create a graph that an NLDR algorithm would not be able to unfold because the shortcut connections would bind geodesically distant regions of the manifold close together.

V. Conclusion

We presented an intelligent neighbor-finding algorithm called SAFFRON, which uses a relaxation technique to select neighbors such that the tangent spaces represented in local neighborhoods will be aligned. This technique enables neighbors to be found that are robust against shortcutting across manifold structure. Because so many important techniques used in machine learning rely on graphs formed by connecting neighboring points, SAFFRON has significant potential to improve the analysis of data that lies on a low-dimensional manifold in high-dimensional space.

The contributions of this paper include: a method for measuring the dihedral angle between two flats with a codimension greater than 1, a method for evaluating the alignment of two tangent spaces by computing the product of the dihedral and monohedral angles, and most significantly, a relaxation technique for determining which points are best-suited to be neighbors, without shortcutting across a manifold structure.

VI. Appendix A

This appendix provides details necessary to implement the last step of the SAFFRON algorithm, which detects and removes shortcut connections with an algorithm called CycleCut [4]. We consider a detailed description of the CycleCut algorithm to be outside the scope of this paper, but we briefly describe its purpose, and we provide pseudocode (see Figure 8) so that this paper will contain all the information necessary to facilitate a complete implementation of SAFFRON.

By definition, a shortcut connection is a neighbor paring between points in geodesically distant regions of a manifold. Shortcut connections necessarily create a large hole in the neighborhood graph. The CycleCut algorithm uses a max-flow/min-cut technique to detect large holes in a graph, and to make a minimal set of cuts to remove them from the graph. In cases where only a small number of shortcut connections exist, CycleCut will detect and remove shortcut connections. Although with many problems SAFFRON is able to construct good neighborhoods that contain no shortcut connections, the addition of CycleCut as a final step to SAFFRON adds extra robustness, such that SAFFRON can be used to handle even more difficult problems.

References

[1] M. Balasubramanian and E. L. Schwartz, "The isomap algorithm and topological stability," *Science*, vol. 295 (5552), 7a, January 2002.

[2] C. Varini, A. Degenhard, and T. W. Nattkemper, "Isolle: LLE with geodesic distance," *Neurocomputing*, vol. 69, no. 13-15, pp. 1768 – 1771, 2006.

[3] E. Levina and P. J. Bickel, "Maximum likelihood estimation of intrinsic dimension," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 777–784.

[4] M. Gashler and T. Martinez, "Robust manifold learning with cyclecut," in *Submission*, 2011.

[5] P. Mahalanobis, "On the generalized distance in statistics," in *Proceedings of the National Institute of Science, Calcutta*, vol. 12, 1936, p. 49.

[6] K. Menger, "Statistical metrics," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 28, no. 12, p. 535, 1942.

[7] S. Stigler, "Francis Galton's account of the invention of correlation," *Statistical Science*, vol. 4, no. 2, pp. 73–79, 1989.

[8] E. Diday, "Recent progress in distance and similarity measures in pattern recognition," in *Second International Joint Conference on Pattern Recognition*, 1974, pp. 534–539.

[9] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research (JAIR)*, vol. 1, pp. 1–34, 1997.

[10] K. B. Jonathan, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *In Int. Conf. on Database Theory*, 1999, pp. 217–235.

[11] K. Nomizu and H. Ozeki, "The existence of complete Riemannian metrics," *Proceedings of the American Mathematical Society*, vol. 12, no. 6, pp. 889–891, 1961.

[12] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf, "Ranking on data manifolds," in *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

[13] J. Wei, H. Peng, Y.-S. Lin, Z.-M. Huang, and J.-B. Wang, "Adaptive neighborhood selection for manifold learning," in *Machine Learning and Cybernetics, 2008 International Conference on*, vol. 1, July 2008, pp. 380–384.

[14] Z. Zhang and H. Zha, "Principal manifolds and nonlinear dimension reduction via local tangent space alignment," *SIAM Journal of Scientific Computing*, vol. 26, pp. 313–338, 2002.

[15] M. Gashler, D. Ventura, and T. Martinez, "Iterative non-linear dimensionality reduction with manifold sculpting," in *Advances in Neural Information Processing Systems 20*. Cambridge, MA: MIT Press, 2008.