Visual Analysis of Dimensionality Reduction Quality for Parameterized Projections

Rafael Messias Martins[a,b], Danilo Coimbra[a,b], Rosane Minghim[a], A. C. Telea[b,c]

[a] *ICMC, University of São Paulo, São Carlos, Brazil, 13566-590*
[b] *Institute Johann Bernoulli, University of Groningen, The Netherlands*
[c] *University of Medicine and Pharmacy 'C. Davila', Bucharest, Romania*

Abstract

In recent years, many dimensionality reduction (DR) algorithms have been proposed for visual analysis of multidimensional data. Given a set of n-dimensional observations, such algorithms create a 2D or 3D projection thereof that preserves relative distances or neighborhoods. The quality of resulting projections is strongly influenced by many choices, such as the DR techniques used and their various parameter settings. Users find it challenging to judge the effectiveness of a projection in maintaining features from the original space and to understand the effect of parameter settings on these results, as well as performing related tasks such as comparing two projections. We present a set of interactive visualizations that aim to help users with these tasks by revealing the quality of a projection and thus allowing inspection of parameter choices for DR algorithms, by observing the effects of these choices on the resulting projection. Our visualizations target questions regarding neighborhoods, such as finding false and missing neighbors and showing how such projection errors depend on algorithm or parameter choices. By using several space-filling techniques, our visualizations scale to large datasets. We apply our visualizations on several recent DR techniques and high-dimensional datasets, showing how they easily offer local detail on point and group neighborhood preservation while relieving users from having to understand technical details of projections.

Keywords: Visual Analytics, Dimensionality Reduction, Parameterization, Projection Errors, Image-based, Large Data

1. Introduction

Dimensionality reduction (DR) techniques are an increasingly popular and pervasive part of visual analytics solutions. Their key value is the ability to transform, or project, high-dimensional datasets into low-dimensional datasets which keep the underlying structure of the data similar. The results can be visualized by scatterplots [1], treemaps, timelines, and parallel coordinates [2]. DR methods have been used for the visual analysis of text documents [3, 4, 5], multimedia [6], text mining [7, 8], vector fields [9], and biomedical data [10, 11].

Although DR techniques have become increasingly more robust and computationally scalable, several major usability challenges still exist. One such challenge involves the quality analysis of DR algorithms. Currently, tens of DR algorithms exist, each with several parameters, whose values strongly influence the projection result. Changes of a single parameter can produce different projections, casting doubt on the correctness or meaning of the resulting projection. However, such parameters are typically quite technical and non-intuitive for the average end-user. Our question is, thus: How to provide insight into the *quality* of DR algorithms, and how to explore their parameter settings, so that users understand how these settings affect the shape, structure, and quality of the resulting projections? In this paper, we present a set of visualization techniques that help users with exploring the link between DR algorithm parameter settings and the quality of the resulting projections. Our visualizations target the following questions:

- How is the projection error spread over the 2D space?
- How to find points which are close in 2D but far in nD?
- How to find points which are close in nD but far in 2D?
- How do DR algorithm choice and parameter settings affect the above quality aspects?

For this, we propose several space-filling techniques that visually scale to large datasets, offer a multiscale (or level-of-detail) view on the projection behavior, and do not require users to understand the internal formulation of DR algorithm. We illustrate our visualizations by exploring the parameters of five state-of-the-art DR techniques for several real-world datasets.

This paper is structured as follows. Section 2 presents related work on DR algorithm quality analysis. Section 3 presents our analysis goals. Section 4 describes our proposed visualizations. Section 5 uses these methods to explore the quality, as function of DR method parameters, of several DR techniques. Section 6 discusses our results. Section 7 concludes the paper.

2. Related Work

2.1. Dimensionality reduction

For a dataset $D^n = \{\mathbf{p}_i \in \mathbb{R}^n\}_{1 \le i \le N}$ of N n-dimensional points, dimensionality reduction (DR) can be seen as a function

$$f : \mathbb{R}^n \times P \to \mathbb{R}^m \tag{1}$$

which maps each point $\mathbf{p}_i \in D^n$ to a point $\mathbf{q}_i \in D^m$. Here, n is typically large (tens up to thousands of dimensions), and m is typically 2 or 3. P denotes the *parameter space* of f, *i.e.* the various settings that control the projection algorithm, including the algorithm type itself. f is designed to keep the so-called *structure* of the data as similar as possible in $\mathbb{R}^n$ and $\mathbb{R}^m$. One way for this is to let f minimize the normalized stress function

$$\sigma = \frac{\sum_{1 \leq i \leq N, 1 \leq j \leq N} (d^n(\mathbf{p}_i, \mathbf{p}_j) - d^m(\mathbf{q}_i, \mathbf{q}_j))^2}{\sum_{1 \leq i \leq N, 1 \leq j \leq N} (d^n(\mathbf{p}_i, \mathbf{p}_j))^2} \tag{2}$$

where $d^n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ and $d^m : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^+$ are distance metrics for D^n and D^m respectively. Other ways to compute f are to optimize for having the k-nearest neighbors for a point $\mathbf{q}_i \in D^m$ be the same as the k-nearest neighbors of $\mathbf{p}_i \in D^n$.

Many DR methods are special cases of a wider class of techniques called Multidimensional Scaling (MDS). MDS methods compute f using only pairwise point distances. This avoids having to access the full nD coordinate data. However, computing distances creates additional costs ($O(N^2)$ for N points). The PLMP algorithm avoids this by using distances only for a small set of representative points and using nD coordinates for the other points [10].

DR methods can be classified by the techniques used to compute f [10]. *Spectral decomposition* techniques project points along the largest-eigenvalue eigenvectors of the pointwise distance matrix [12]. LLE [13] and ISOMAP [14, 15] use efficient numerical methods tailored to solve sparse eigenproblems. Landmarks MDS [16] and Pivot MDS [17] book further speed-ups by using classical MDS on a subset of representative points and projecting remaining points by local interpolation. Fastmap achieves linear complexity in the input point count but has a worse stress minimization [6].

Nonlinear optimization methods iteratively search the parameter space P to minimize the stress σ [18, 19]. Besides naive gradient descent, multigrid numerical solvers can be used to speed searching [20]. Pekalska *et al.* propose a speed-up that projects a representative subset (by gradient descent) and fits remaining points by local interpolation [21]. Force-based methods are a special class of nonlinear optimization with many uses in graph drawing [22]. Chalmers speeds this up by using the representative subset idea outlined earlier [23]. Further speed-ups are achieved by multilevel solvers and GPU techniques [24, 25], and by recursively selecting representatives via a multilevel approach [26]. Tejada *et al.* use a heuristic to embed instances by force-based relaxation [27]. LSP positions the representative subset by a force-based scheme and fits the remaining points by Laplacian smoothing [4]. LAMP also uses a representative subset to locally construct affine projections, and allows users to interactively place these points to optimize the overall projection layout [3]. More details on LSP, LAMP, and ISOMAP are given further in Section 5.2.

2.2. Visualizing projection quality

Although projection quality is acknowledged as important, most DR literature considers mainly aggregated quality metrics such as the stress function (Eqn. 2), correlation [28], neighborhood preservation average plots [4], and distance scatterplots [3], which are distance and neighborhood based metrics, or cluster segregation metrics [29]. 2D scatterplots can show the correlation of D^n with D^m [3]. Such metrics capture the *overall* quality of a projection, but do not help finding local quality variations. In other words, they do not show projection problems for *any* point i *vs all* points $j \neq i$ in the input dataset.

Local metrics can be used to highlight where (in a projection) errors happen. Shreck *et al.* compute, for each $\mathbf{p} \in D^n$, the projection precision score (*pps*) defined as the normalized distance between the two k-dimensional vectors having as components the Euclidean distances between $\mathbf{p}$ and its k nearest neighbors in D^n, respectively D^2 [30]. Visualizing *pps* as a color map shows areas where neighborhoods are not preserved. However, a neighborhood cannot be preserved for two distinct reasons: true neighbors (in D^n) are missing (in D^2), or neighbors (in D^2) are actually false neighbors (in D^n). The *pps* metric does not differentiate between such situations, and can also be sensitive to permutations of points that do not change distances.

Recognizing that DR methods can create distance approximation errors, Van der Maaten *et al.* extend the t-SNE technique [31] to output a set $\{M_i\}$ of 2D projections rather than a single one [32]. All points appear in all projections M_i, with potentially different weights and at different locations. This allows better modeling non-metric similarities. Yet, correlating points over the several M_i is done manually by the user, and can be challenging for large datasets and many projections M_i.

Several quality metrics for continuous DR techniques are proposed by Aupetit [33]. Point-based stretching and compression metrics measure, for each $\mathbf{p}_i \in D^n$, the aggregated increase, respectively decrease, of the distances of its projection $\mathbf{q}_i \in D^2$ to all other projections $\mathbf{q}_{j \neq i}$ *vs* the distances of $\mathbf{p}_i$ to all other points $\mathbf{p}_{j \neq i}$. Segment stretching and compression measures the variation of distances of close point pairs (i, j) between $\mathbb{R}^n$ and $\mathbb{R}^2$. For a selected $\mathbf{p}_i$, the proximity metric maps distances in $\mathbb{R}^n$ from $\mathbf{p}_i$ to all other points $\mathbf{p}_{j \neq i}$ to the corresponding points $\mathbf{q}_i \in \mathbb{R}^2$ and thereby helps understanding how (and where) the projection may have distorted the structure of the data. These metrics are visualized with piecewise-constant interpolation of the point, respectively segment, data using Voronoi diagrams. Our proposed techniques in Secs. 4.2, 4.3, and 4.4 adapt and extend these visualizations in several directions.

Still using colored Voronoi cells, Lespinats and Aupetit show, at the same time, point stretching and compression by using a 2D color map [34]. The proposed color map encodes stretching as green, compression as purple, low-error points as white, and points with high stretching and compression as black, respectively. While this color map can show local error types (or the absence thereof), it cannot explicitly show the point-pairs which cause stretching and compression. Besides, as the authors also note, Voronoi cells can lead to visualization bias due to the cells' sizes and shapes being heavily dependent on the D^2 point density, and the fact that cells cover the entire $\mathbb{R}^2$ space, even in areas where no projected points exist.

To assist the task of navigating projections while also considering distortions, Heulot *et al.* present an interactive semantic lens that filters points projected too closely to a user-selected focus point in $\mathbb{R}^2$ [35]. Such points, also called false neighbors, are pushed towards the lens border, so they do not attract the user's attention. Separately, points are colored by the distance in D^n to the focus point, to help users navigate to the so-called missing neighbors of the focus point. Instead of Voronoi cells of [33, 34], points are colored using Shepard interpolation, which yields a smoother, and arguably less distracting, image. However, in contrast to [33, 34], this method can only show errors related to a selected focus point.

3. Analysis goals

A projection f should preserve the *structure* of the original space $\mathbf{R}^n$. This implies a mix of distance and neighborhood preservations at various scales and happens at different rates for different datasets, projection algorithms and parameter values. For users, the projection's *precision* [30] is not clear unless they can interpret projected neighborhoods adequately [33]. Thus, given any DR algorithm (Eqn. 1), we aim to show how neighborhood preservation is affected by choices of parameter values in P, highlighting aspects that can adversely affect the interpretation of the projected point set in D^m. To simplify the discourse, we next consider $m = 2$, and that projections are drawn as scatterplots (the most common option for DR visualization). We identify the following aspects of interest:

A. False neighbors: Take a point $\mathbf{p}_i \in D^n$ and its 2D projection $\mathbf{q}_i = f(\mathbf{p}_i)$. A necessary condition for neighborhood preservation is that *all* points $\mathbf{q}_j$ which are close to $\mathbf{q}_i$ (in 2D) should be projections of points $\mathbf{p}_j$ which are close to $\mathbf{p}_i$ (in D^n). If not, *i.e.* we have a $\mathbf{q}_j$ close to $\mathbf{q}_i$ for which $\mathbf{p}_j$ is not close to $\mathbf{p}_i$, the user wrongly infers from the projection that $\mathbf{p}_j$ is close to $\mathbf{p}_i$. We call such a point j a *false neighbor* of i.

B. Missing neighbors: The second necessary condition for neighborhood preservation is that *all* $\mathbf{p}_j$ which are close to $\mathbf{p}_i$ (in D^n) project to points $\mathbf{q}_j$ which are close to $\mathbf{q}_i$ (in 2D). If not, *i.e.* we have a $\mathbf{p}_j$ close to $\mathbf{p}_i$ for which $\mathbf{q}_j$ is not close to $\mathbf{q}_i$, the user will underestimate the set of points similar to point i. We call such a point j a *missing neighbor* of i.

C. Groups: A main goal of DR is to help users find groups of similar points, *e.g.* topics in a document set [3, 4] or classes of images in a database [6]. False and missing point neighbors generalize, for groups, to *false members* and *missing members* respectively. Given a group Γ of closely projected points, we aim to find if *all* points in Γ truly belong there (no false members), and if *all* points that belong to the topic described by Γ do indeed project in Γ (no missing members).

D. Detail: Aggregated local metrics such as [30, 33, 34, 35] can show, up to various extents, where missing or false neighbors occur. However, they do not directly show which are all such neighbors, for each projected point. Also, they do not explicitly address locating false and missing group members. We aim to provide interactive visual mechanisms to support these tasks on several levels of detail.

4. Visualization methods

We next propose several visualization methods to address the analysis goals outlined in Sec. 3. As a running example, we use LAMP as projection method, with the default parameter settings given in [3], and as input the well-known 19-dimensional Segmentation dataset with 2300 points from [36, 3, 37, 10]. Herein, each point describes a randomly drawn 3x3 pixel-block from a set of 7 manually segmented outdoor images, by means of 19 statistical image attributes, such as color mean, standard deviation, and horizontal and vertical contrast.

4.1. Preliminaries

To quantify the neighborhood preservation issues in Sec. 3, we first define the projection error of point i *vs* a point $j \neq i$ as

$$e_{ij} = \frac{d^m(\mathbf{q}_i, \mathbf{q}_j)}{\max_{i,j} d^m(\mathbf{q}_i, \mathbf{q}_j)} - \frac{d^n(\mathbf{p}_i, \mathbf{p}_j)}{\max_{i,j} d^n(\mathbf{p}_i, \mathbf{p}_j)}. \quad (3)$$

We see that $e_{ij} \in [-1, 1]$. Negative errors indicate points whose projections are too close (thus, false neighbors). Positive errors indicate points whose projections are too far apart (thus, missing neighbors). Zero values indicate 'good' projections, which approximate optimally the distances in D^n.

4.2. Aggregated error view

We first provide an overview of how the projection error spreads over an entire dataset, by computing for each point i the aggregate error

$$e_i^{aggr} = \sum_{j \neq i} |e_{ij}|. \quad (4)$$

The value of e_i^{aggr} gives the projection error of point i with respect to *all* other points. Low values of e^{aggr} show points whose projections can be reliably compared with most other projections in terms of assessing similarity. These are good candidates for representatives in multilevel projection methods [6, 21, 23, 4]. Large values of e^{aggr} show points which are badly placed with respect to most other points. These are good candidates for manual projection optimization [38, 37].

Fig. 1 (a) shows e^{aggr} by color mapping its value on the 2D projected points, using a blue-yellow-red diverging colormap [39]. Brushing and zooming this image allows inspecting e^{aggr} for individual points. However, given our goal of providing an overview first, we are actually not interested in *all* individual e^{aggr} values, but rather to (a) find compact areas in the projection having similar e^{aggr} values, (b) find outlier e^{aggr} values in these areas (if any), and (c) see how e^{aggr} globally varies across the projection. For this, we propose an image-based, space-filling visualization, as follows. Denote by $DT(\mathbf{x} \in \mathbb{R}^2) = \min_{\mathbf{q} \in D^m} \|\mathbf{q} - \mathbf{x}\|$ the so-called distance transform of the 2D point cloud D^m delivering, for any screen pixel $\mathbf{x}$, its distance to the closest point in D^m. We then compute e^{aggr} at every screen pixel $\mathbf{x}$ as

$$e^{aggr}(\mathbf{x}) = \frac{\sum_{\mathbf{q} \in N_\epsilon(\mathbf{x})} \exp\left(-\frac{\|\mathbf{x}-\mathbf{q}\|^2}{\epsilon^2}\right) e^{aggr}}{\sum_{\mathbf{q} \in N_\epsilon(\mathbf{x})} \exp\left(-\frac{\|\mathbf{x}-\mathbf{q}\|^2}{\epsilon^2}\right)} \quad (5)$$

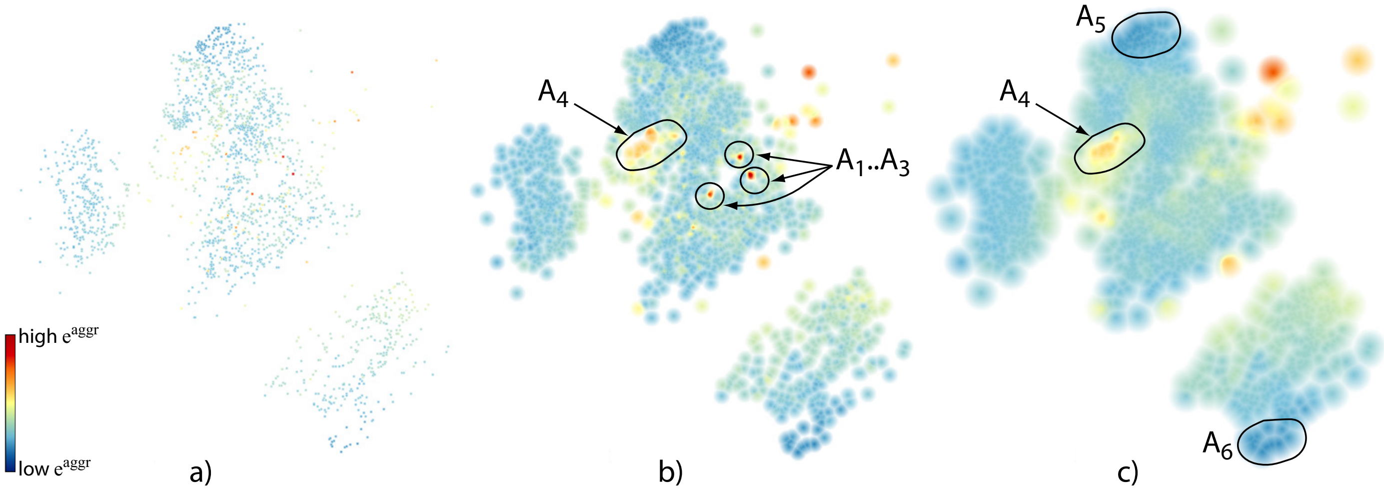

Figure 1: Aggregate error view, several levels of detail: (a) $\alpha = 1, \beta = 1$. (b) $\alpha = 5, \beta = 5$. (c) $\alpha = 20, \beta = 20$ pixels (see Sec. 4.2).

with

$$\epsilon = DT(\mathbf{x}) + \alpha. \tag{6}$$

Here, $N_\epsilon(\mathbf{x})$ contains all projections in D^m located within a radius ϵ from $\mathbf{x}$. We next draw $e^{aggr}(\mathbf{x})$ as a RGBA texture, where the color components encode $e^{aggr}(\mathbf{x})$ mapped via a suitable color map, and the transparency A is set to

$$A^{aggr}(\mathbf{x}) = \begin{cases} 1 - \frac{DT(\mathbf{x})}{\alpha}, & \text{if } DT(\mathbf{x}) < \beta \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

For $\alpha = 1, \beta = 1$, we obtain the classical colored scatterplot (Fig. 1 (a)). For $\alpha = 1, \beta > 1$, the space between projections is filled, up to a distance β, by the e^{aggr} value of the closest data point. For $\alpha = 1, \beta = \infty$, we obtain a Voronoi diagram of the projections with cells colored by their e^{aggr} values. This does not change the e^{aggr} data values, but just displays them on larger spatial extents than individual pixels, making them easier to see. This creates visualizations identical to those obtained by drawing scatterplots with point radii equal to β, without having the issues created by overlapping points. For $\alpha > 1, \beta > 1$, the result is similar to Shepard interpolation where the kernel size ϵ is given by the *local* point density. The parameter $\alpha \geq 0$ controls the *global* level-of-detail at which we visualize e^{aggr}: Small values show more detail in dense point zones, but also emphasize small-scale signal variations which are less interesting. Larger α values create a smoother signal where coarse-scale error patterns are more easily visible.

Figs. 1 (b,c) show the aggregate error for the Segmentation dataset for various values of the parameters α and β. Here, $e_{ij} \in [-0.67, 0.35]$. The error range already tells that we have poorly projected points, but does not tell where these are. In Fig. 1 (b), with low values for both α and β, we see that e^{aggr} is relatively smoothly distributed over the entire projection. However, we see three small red spots $A_1..A_3$. These are high-error outlier areas, which indicate points that are badly placed with respect to *most* other points. We also see a relatively high error area A_4 of larger spatial extent. Increasing both α and β produces a simplified visualization (Fig. 1 (c)). Larger β values fill in the gaps between points. Larger α values eliminate outlier regions whose spatial extent is smaller than α, such as the three small outlier areas $A_1..A_3$, but A_4 remains visible, since it is larger than α. We now also notice, better than in Fig. 1 (b), that the bottom and top areas (A_5, A_6) in the projection have dark blue values, with a significantly lower error than the rest of the projection.

Our image-based results are slightly reminiscent of the dense *pps* maps of Schreck *et al.* [30] (see Sec. 2.2). Differences exist, however. First, our e_i^{aggr} is a *global* metric, that tells how point i is placed with respect to all other points, whereas the *pps* metric characterizes *local* neighborhoods. Interpolation-wise, our technique (used with $\alpha = 1, \beta = \infty$) delivers the same Voronoi diagram as Schreck *et al.*, which is also identical to the space partitioning of the point-based Voronoi diagrams in [33, 34]. The data being mapped is, however, different: Our e^{aggr} shows the sum of distance compression and stretching, whereas [33, 34] treat these two quantities separately. In the next sections, we show how we split our aggregated insight into separate insights. Further on, both Schreck *et al.* and our method use smoothing to remove small-scale noise from such maps. However, whereas Schreck *et al.* uses a constant-radius smoothing kernel, which blurs the image equally strong everywhere, we use, as explained, a variable-radius kernel controlled by local density, which preserves better detail in non-uniform point clouds.

4.3. False neighbors view

However useful to assess the error distribution and find badly *vs* well-projected point groups, the aggregate error view does not tell us if the error is due to false neighbors, missing neighbors, or both. Let us first consider the false neighbors (case **A**, Sec. 3). To visualize these, we create a Delaunay triangulation of the projected point cloud that gives us the closest neighbors of each projected point in all directions, *i.e.*, the most important false-neighbor candidates for that point. To each edge $E_k, 1 \leq k \leq 3$ of each triangle T of this triangulation, with vertices being the points $\mathbf{q}_i$ and $\mathbf{q}_j$ of D^m, we assign a weight

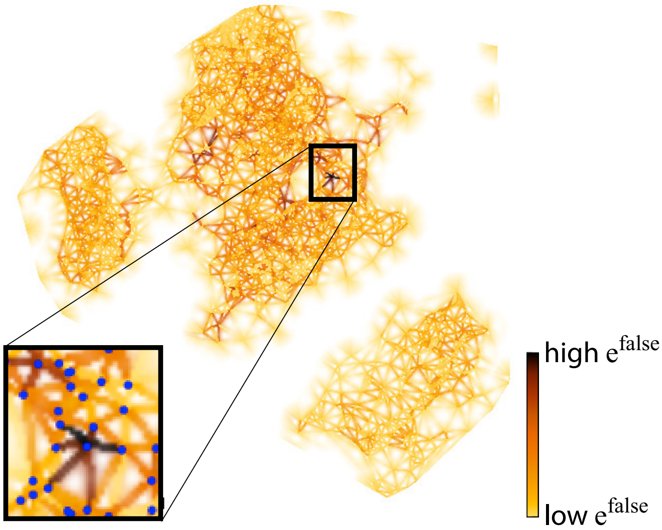

Figure 2: False neighbors view (see Sec. 4.3).

$e_k^{false} = |\min(e_{ij}, 0)|$, *i.e.*, consider only errors created by false neighbors. Next, we interpolate e^{false} over all pixels $\mathbf{x}$ of T by using

$$e^{false}(\mathbf{x}) = \frac{\sum_{1 \le k \le 3} \frac{1}{d(\mathbf{x}, E_k)\ \|E_k\|} e_k^{false}}{\sum_{1 \le k \le 3} \frac{1}{d(\mathbf{x}, E_k)\ \|E_k\|}} \tag{8}$$

where $d(\mathbf{x}, E)$ is the distance from $\mathbf{x}$ to the edge E and $\|E\|$ is the length of the edge. Similarly to the aggregated error, we construct and render an image-based view for e^{false} as a RGBA texture. In contrast to the aggregated error, we use here a heated body colormap [39], with light hues showing low e^{false} values and dark hues showing high e^{false} values. This attracts the attention to the latter values, while pushing the former ones into the background. The transparency A is given by

$$A^{false}(\mathbf{x}) = A^{aggr}(\mathbf{x}) \left(1 - \frac{1}{2}\left(\min\left(\frac{DT_T(\mathbf{x})}{DT_C(\mathbf{x})}, 1\right) + \max\left(1 - \frac{DT_C(\mathbf{x})}{DT_T(\mathbf{x})}, 0\right)\right)\right) \tag{9}$$

where $DT_T(\mathbf{x}) = \min(d(\mathbf{x}, E_1), d(\mathbf{x}, E_2), d(\mathbf{x}, E_3))$ is the distance transform of T at $\mathbf{x}$, $DT_C(\mathbf{x})$ is the distance from $\mathbf{x}$ to the barycenter of T, and A^{aggr} is given by Eqn. 7. The same technique is used in a different context to smoothly interpolate between two 2D nested shapes [40], where we refer for further implementation details. The combined effect of Eqns. 8 and 9 is to slightly thicken, or smooth out, the rendering of the Delaunay triangulation. Note that this interpolation does *not* change the actual values e_k^{false} rendered on the triangulation edges. The distance-dependent transparency ensures that data is shown only close to the projection points.

Fig. 2 shows the false neighbors for the Segmentation dataset. Several things are apparent here. First, the rendering is similar to a blurred rendering of the Delaunay triangulation of the 2D projections colored by e^{false}, showing how each point relates to its immediate neighbors. Light-colored edges show true neighbors, while dark edges show false neighbors. Since edges are individually visible, due to the transparency modulation (Eqn. 9), we can see both the true and false neighbors of a point separately. The smooth transition between opaque points (on the Delaunay edges) and fully transparent points (at the triangles' barycenters) ensures that the resulting image is continuous and easier to follow at various screen resolutions than a Delaunay triangulation rendered with pixel-thin edges, as our edges appear slightly thicker.

In Fig. 2, two error-related aspects are visible. First, we see an overall trend from light to dark colors as we go further from the projection's border towards the projection center. This confirms the known observation on DR methods that projections on the border tend to be more accurate, since there is more freedom (and space) to place these. In contrast, projections falling deep inside the resulting point cloud tend to have more false neighbors, because the DR algorithm has there less space to shift points around to accommodate all existing distance constraints. Intuitively, we can think of this phenomenon as a 'pressure' which builds up within the projected point set from its border inwards. We shall see more examples of this phenomenon in Sec. 5. Secondly, we see a few small-scale dark outliers. Zooming in Fig. 2, we see that these are points connected by dark edges to most of their closest neighbors in a star-like pattern. Clearly, false neighbors exist here. These can be either the star 'center' or the tips of its branches. However, we also see that these tips have only one dark edge. Hence, they are too closely positioned to the star center only, and not to their other neighbors. Since the tip points are all positioned well with respect to their neighbors (except the star center), and the center point is positioned too closely with respect to all its direct neighbors, we can conclude that too little space was offered in the projection to the center point, or in other words that the center point is a false neighbor of its surrounding points.

The false neighbors view is related to Aupetit's segment compression view, where the shortening of inter-point distances due to projection is visualized [33]. The underlying metrics, *i.e.* our e_{ij} (Eqn. 3) and m_{ij}^{distor} ([33], Sec. 3.2) are similar, up to

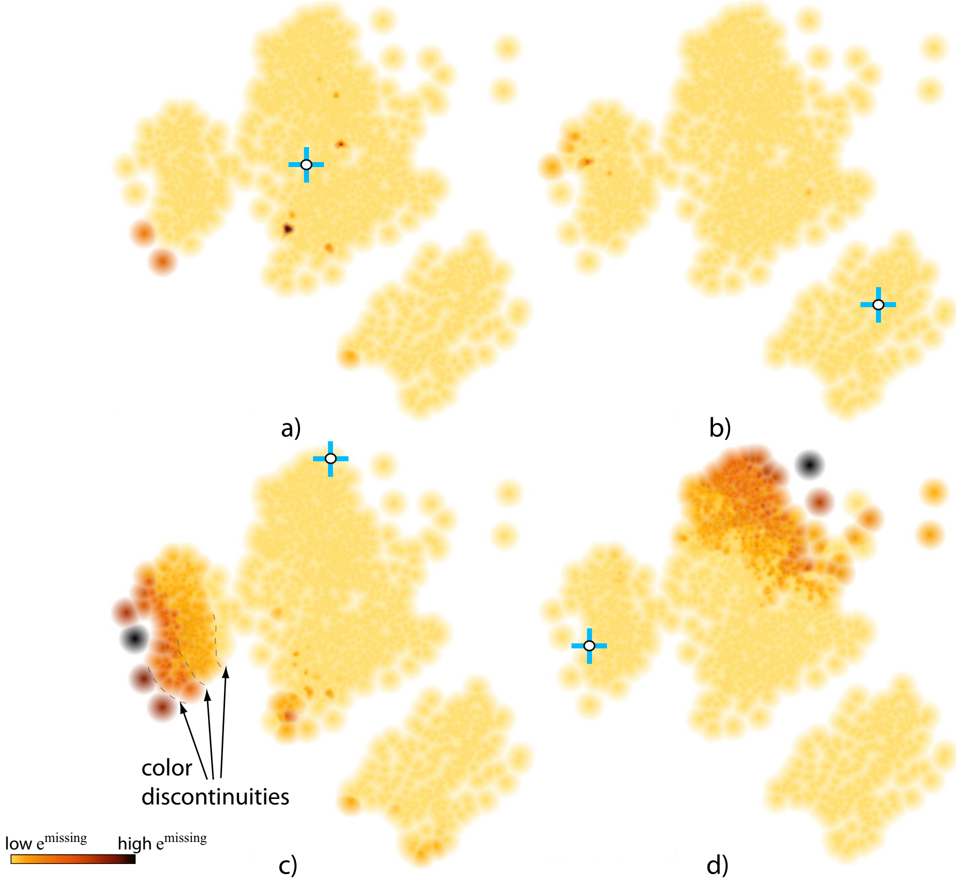

Figure 3: Missing neighbors view for different selected points. Selections are indicated by markers (see Sec. 4.4).

different normalizations. However, the proposed visualizations are quite different. Aupetit uses so-called 'segment Voronoi cells' (SVCs). SVCs essentially achieve piecewise-constant interpolation of the values e_k^{false}, defined on the edges E_k of each Delaunay triangle T, over T's area, by splitting T in three sub-triangles using its barycenter. In contrast, our interpolation (Eqn. 8) is C^∞ over T. Also, our triangles are increasingly transparent far away from their edges (Eqn. 9). Comparing our results (*e.g.* Figs. 2, 9 (a,d,g)) with SVCs (*e.g.* Figs. 7 (d), 12 (c) in [33]), we observe that SVCs exhibit several spurious elongated Voronoi cells that do not convey any information. Such cells do not exist in our visualization due to the transparency blending. Also, we argue that the artificial SVC edges linking projected points with Delaunay triangulation barycenters do not convey any information, but only make the visualization more complex. Such edges do not exist in our visualization due to our continuous interpolation.

4.4. Missing neighbors view

Besides false neighbors, projection errors (and subsequent misinterpretations) can also be caused by missing neighbors (case **B**, Sec. 3). Visualizing this by a space-filling method like for the aggregate error or false neighbors is, however, less easy. Given a projected point $\mathbf{q}$, its missing neighbors can be anywhere in the projection, and are actually by definition far away from $\mathbf{q}$. To locate such neighbors, we would need to visualize a many-to-many relation between far-away projected points.

We first address this goal by restraining the question's scope: Given a *single* point $\mathbf{q}_i$, show which of the other points $D^m \setminus \mathbf{q}_i$ are missing neighbors for $\mathbf{q}_i$. For this, we first let the user select $\mathbf{q}_i$ by means of direct brushing in the visualization. Next, we compute the error $e_i^{missing} = \max_{j \neq i}(e_{ij}, 0)$, *i.e.*, the degree to which $\mathbf{q}_j$ is a missing neighbor for $\mathbf{q}_i$, and visualize $e^{missing}$ by the same technique as for the aggregated error (Sec. 4.2).

Fig. 3 shows this for the Segmentation dataset, using the same heat colormap as in Fig. 2. In Figs 3 (a,b), we selected two points deep inside the central, respectively the lower-right point groups in the image. Since Figs. 3 (a,b) are nearly entirely light-colored, it means that these points have few missing neighbors. Hence, the 2D neighbors of the selected points are truly *all* the neighbors that these points have in nD. In Figs. 3 (c,d), we next select two points located close to the upper border of the large central group and the left border of the left group respectively. In contrast to Figs. 3 (a,b), we see now an increasingly darker color gradient as we go further from the selected points. This shows that points far away from these selections are actually projected *too* far, as they are actually more similar than the projection suggests. This is a known (but never visualized as such) issue of many DR methods, which have trouble in embedding high-dimensional manifolds in 2D: points close to the embedding's *border* are too far away from other points in the projection. Another interesting finding is that the color-coded Figs. 3 (c,d) do not show a *smooth* color gradient: We see, es-

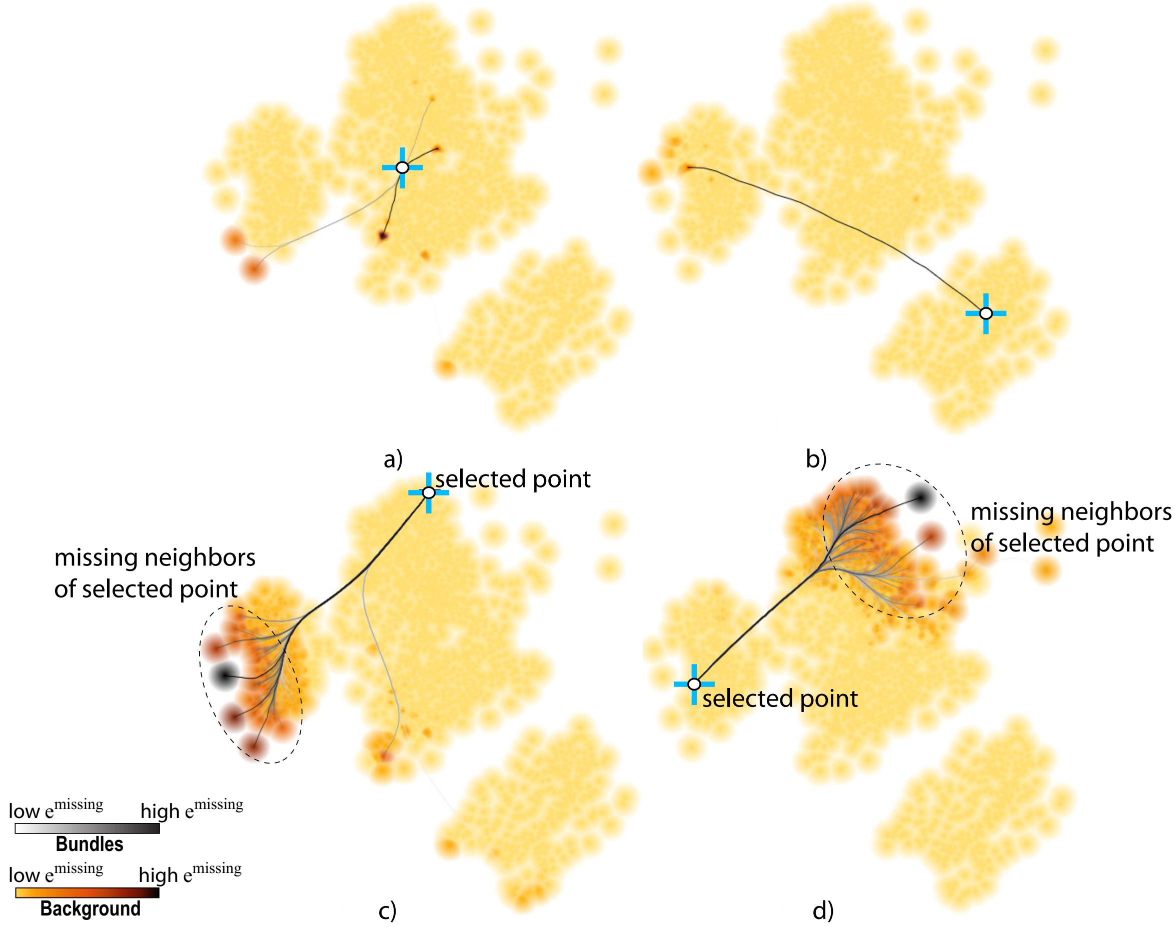

Figure 4: Missing neighbors finder view for four selected points. Selections are indicated by markers (see Sec. 4.5).

pecially in Fig. 3 (c) that the colors appear grouped in several 'bands', separated by discontinuities. In other words, the projection method suddenly increases the error as we get over a certain maximal 2D distance.

The missing neighbors view is related to the proximity view of Aupetit [33]. In both views, a point i is selected and a scalar value, related to this selection, is plotted at all other points $j \neq i$. For Aupetit, this is the distance $m_j^{prox} = d^n(\mathbf{p}_i - \mathbf{p}_j)$ (normalized by its maximum). For us, it is the error $e_j^{missing}$. Both the distance and $e^{missing}$ have, in general, the tendency to be small at points j close in 2D to the selected point i, and increase farther off from point i. However, the two quantities are different and serve different purposes. Visualizing m^{prox} is useful in finding points located within some distance to the selection i. Finding projection errors is only *implicitly* supported, as these appear as non-monotonic variations in the m^{prox} signal. In contrast, $e^{missing}$ specifically emphasizes points projected too far, rather than conveying the absolute distance. Thus, our visualization helps locating projection errors rather than assessing proximity.

4.5. Missing neighbors finder

Although providing details for single points, the views in Sec. 4.4 cannot show missing neighbors for an entire dataset. We address this goal by a different method, as follows. Consider all positive values of e_{ij}. By definition, these give all point-pairs which are projected too far away. We sort these values decreasingly, and select the largest ϕ percent of them, where ϕ is a user-provided value. The selected values give the point pairs which are worst placed in terms of overestimating their true similarity. We next construct a graph $G = (V, E)$ whose nodes V are the projected points $\mathbf{q}_i$ present in such point pairs, and edges E indicate the pairs, with e_{ij} added as edge weights. Next, we draw G using the KDEEB edge bundling technique [41], which provides robust, easy to use, and real-time bundling of graphs with tens of thousands of edges on a modern GPU. We color the bundled edges based on their weight using a grayscale colormap (with white mapping low and black mapping high weights), and draw them sorted back-to-front on weight and with an opacity proportional to the same weight. The most important edges thus appear atop and opaque, and the least important ones are at the bottom and transparent.

Fig. 4 shows this visualization, which we call the missing neighbors finder, with bundles that connect a single selected point with its most important missing neighbors (bundles connecting multiple points are discussed later on). The background images show $e^{missing}$ (Sec. 4.4). Dark bundle edges attract attention to the most important missing neighbors. For the selected points in images (a) and (b), we see that there are only very few and unimportant missing neighbors (few half-transparent edges). For the selected points in images (c) and (d), the situa-

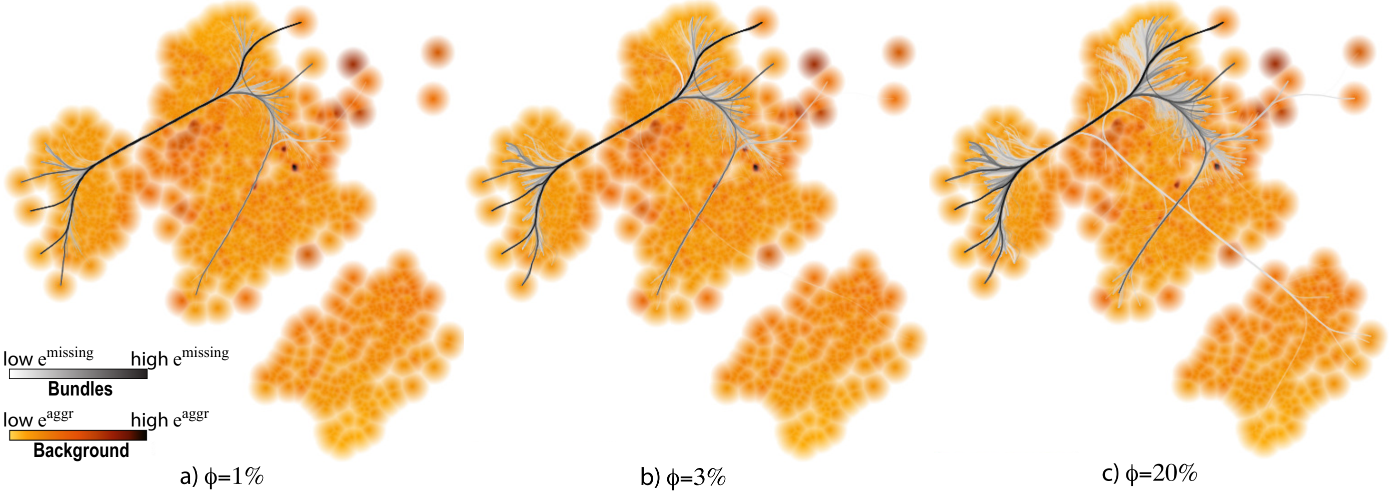

Figure 5: Missing neighbors finder view, all point pairs, for different ϕ values (see Sec. 4.5).

tion is different, as the bundles are thicker and darker. Bundle fanning shows the *spread* of missing neighbors for the selected points: In image (c), these are found mainly in the left point group, with a few also present in the lower part of the central group. In contrast, all missing neighbors of the point selected in image (d) are at the top of the central group.

The main added value of the missing neighbors finder appears when we visualize the many-to-many relations given by all projected points. Fig. 5 shows this result for three values of ϕ for the Segmentation dataset. The background shows now the aggregated error (e^{aggr}, Sec. 4.2). We color bundles from black for largest error e_{ij} to white for largest error above the user-provided parameter ϕ. Image (a) shows the $\phi = 1\%$ worst missing-neighbor point-pairs. These link the top-right area of the central group with the left frontier of the left group. Adding more missing neighbor pairs to the view (image (b), $\phi = 3\%$) strengthens this impression. Adding even more missing neighbor pairs (image (c), $\phi = 20\%$) reveals additional missing-neighbor pairs between the two areas indicated above (light gray parts of thick top bundle), and also brings in a few missing neighbors between these areas and the lower-right point group (light gray thin bundle going to this group). Nearly all bundles appear to connect point pairs located on the *borders* of the projection. This strengthens our hypothesis that such point pairs are challenging for the LAMP projection, which we noticed using the interactive missing neighbors view (Sec. 4.4). However, as compared to that view, the bundled view shows all such point pairs in a single go, without requiring user interaction.

4.6. Group analysis views

As outlined in Sec. 3, the false and missing neighbors issues for individual points become, at group level, the problems of false and missing group members respectively. We next propose two visualizations that assist in finding such issues.

First, let us refine the notion of a group. Given the tasks in Sec. 3 (C), a group $\Gamma \subset D^m$ is a set of projected points which form a *visually* well-separated entity. When users see points in a group, they understand that these share some commonality, but are different from points in other groups. In the LAMP projection of our Segmentation dataset, we see three such groups (Figs. 1-5). Group perception is, obviously, subject to many factors such as user preferences and level-of-detail at which one focuses. However, once a user has established which are the groups (s)he sees in a visualization, the false and missing membership issues become relevant.

We allow users to select groups in a given projection by several mechanisms: direct interactive selection, mean-shift clustering [42], and upper thresholding of the point density [43]. Other user-controlled methods can be used if desired, *e.g.*, *K*-means or hierarchical agglomerative clustering *e.g.* [44, 45]. The actual group selection mechanism is further of no importance to our visualization method. We next render each obtained group $\Gamma = \{\mathbf{q}_i\}$ by the shaded cushion technique in [46] as follows. First, we compute a density map $\rho(\mathbf{x}) = \sum_{\mathbf{q}\in\Gamma} K(\mathbf{x}-\mathbf{y})$, where K is an Epanechnikov kernel of width equal to the average inter-point distance δ in Γ, following [42]. Next, we compute a threshold-set Γ_δ of ρ at level δ, and its distance transform DT_{Γ_δ}. Finally, we render a RGBA texture over Γ_δ, where we set the color a fixed hue (light blue in our case) and the transparency A to $\sqrt{DT_{\Gamma_\delta}}$.

Having now groups both as a data structure and also shown in the visualization, we adapt the missing neighbors and finder techniques (Secs. 4.4, 4.5) to show missing group members. For this, we compute a value

$$e_\Gamma^{missing}(\mathbf{q}_i) = \begin{cases} \min_{\mathbf{q}_j\in\Gamma}(e_{ij}) & \text{if } \mathbf{q}_i \notin \Gamma \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

at each projected point $\mathbf{q}_i$, and visualize $e_\Gamma^{missing}$ using the same technique as for missing neighbors.

Fig. 6 (a,b) show two missing group members views. The shaded cushions show the three groups identified in our Segmentation dataset. Several points fall outside of all groups. This is normal, in general, *e.g.* when the user cannot decide to which group to associate a point. In image (a), we select the bottom group Γ_{bottom}. The underlying color map shows now

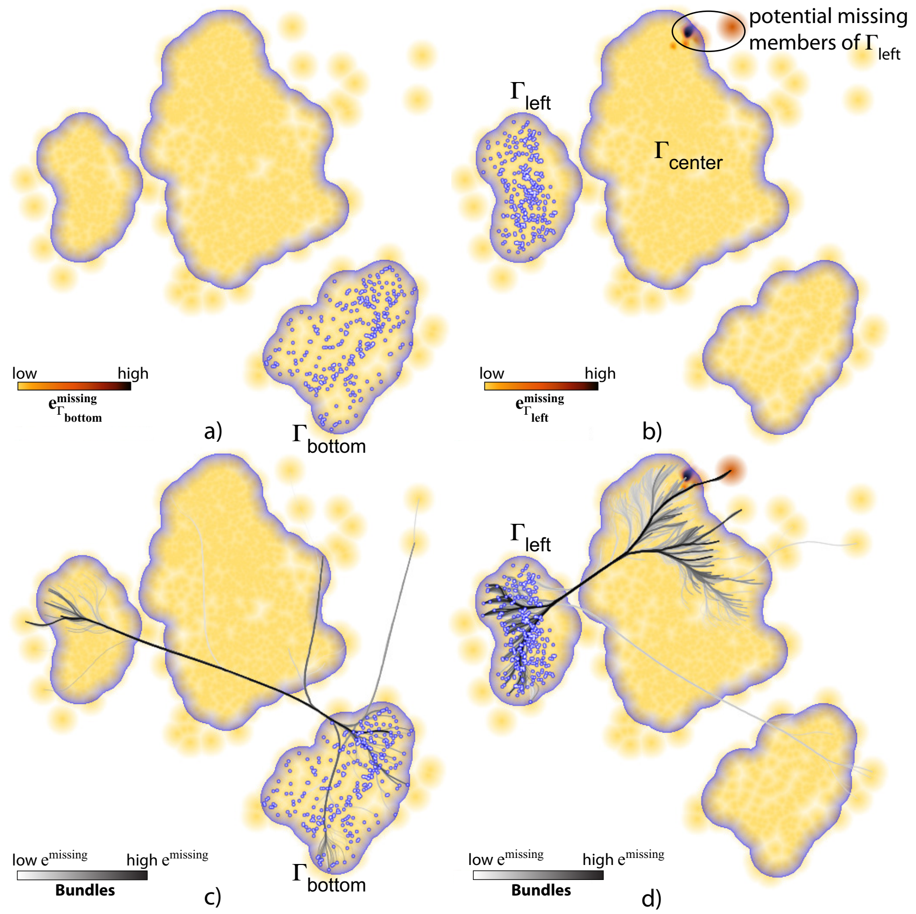

Figure 6: Missing members for two point groups. Points in the selected groups are drawn as marked (see Sec. 4.6).

$e^{missing}_{\Gamma_{bottom}}$, (Eqn. 10). All points appear light yellow. This means that, with respect to Γ_{bottom} *seen as a whole*, no points are projected too far, so Γ_{bottom} has no missing members. In image (b), we do the same for the left group Γ_{left}. The image now appears overall light yellow, except for a small dark-red spot in the upper-right corner of the central group Γ_{center}. Here are a few points which are placed too far from any point in Γ_{left}. These are highly likely to be missing members of Γ_{left}. To obtain more insight, we now use the bundle view in Sec. 4.5, with two changes. First, we build only bundles that have an endpoint in the selected group. Secondly, we consider all edges rather than showing only the most important ones. Image (c) shows the bundle view for Γ_{bottom}. We see only a few bundled edges, ending at a small subset of the points in Γ_{bottom}. This strengthens our hypothesis that there are no points outside Γ_{bottom} which should be placed closer to *all* points in Γ_{bottom} – or, in other words, that Γ_{bottom} has no missing members. Image (d) shows the bundled view for Γ_{left}. The bundle structure tells us that the top-right part of Γ_{center} contains many missing neighbors of Γ_{left}. In particular, we see dark bundle edges that connect to dark-red points. This is a strong indication that these points can indeed be missing members of Γ_{left}. For a final assessment, the user can interactively query the discovered points' details (attribute values) and, depending on these, finally decide if these points are missing group members or not.

4.7. Projection comparison view

Consider running the same DR algorithm with two different parameter sets, or projecting a dataset by two different DR algorithms. How to compare the results from the viewpoint of neighborhood preservation? Subsequent questions are: Which points that were (correctly) placed close to each other in one projection are now 'pulled apart' in the other projection? Do the two projections deliver the same groups of points?

To answer such questions, we propose the *projection comparison* view. The view reads two projections D^m_1 and D^m_2 of the same input dataset D^n. For each point-pair ($\mathbf{q}^1_i \in D^m_1, \mathbf{q}^2_i \in D^m_2$), we compute a displacement

$$e^{disp}_i = \frac{\|\mathbf{q}^1_i - \mathbf{q}^2_i\|}{\max_i \|\mathbf{q}^1_i - \mathbf{q}^2_i\|}. \tag{11}$$

We next build a graph whose nodes are points in $D^m_1 \cup D^m_2$. Edges relate point pairs ($\mathbf{q}^1_i \in D^m_1, \mathbf{q}^2_i \in D^m_2$), and have the values e^{disp} as weights. We visualize this graph via edge bundling, as for the missing neighbors finder (Sec. 4.5).

Fig. 7 (a) shows a view where we compare the Segmentation dataset projected via LAMP (red points, D^m_1) and LSP (green points, D^m_2). The two projections are quite similar, since red and green points occur together in most cases. However, this image does not tell if the two projections create the same *groups* of points, since we do not know how red points match the green

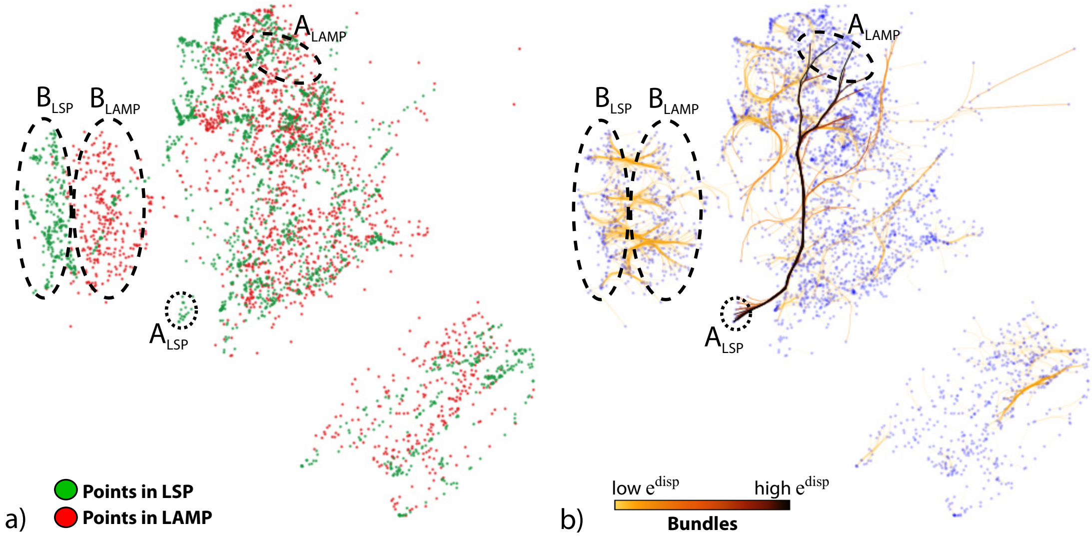

Figure 7: Comparison of two projections. (a) LAMP (blue) and LSP (red) points. (b) Bundles show corresponding point groups in the two projections (see Sec. 4.7).

ones. Fig. 7 (b) shows the projection comparison view for this case. We immediately see a thin dark bundle in the center: This links corresponding points which differ the most in the two projections. Correlating this with image (a), we see that LSP decided to place the respective points at the bottom (A_{LSP}) of the central group, while LAMP moved *and* also spread out these points to the top (A_{LAMP}). However, points around the locations A_{LSP} and A_{LAMP} do not move much between the two projections, as we see only light-colored bundles around these locations, apart from the dark bundle already discussed. Hence, the motion of these points indicates a neighborhood problem in one or both of the projections. Indeed, if *e.g.* the points in A were correctly placed by LAMP (into A_{LAMP}), then the decision of LSP to move the point-group A all the way up in the visualization (to A_{LSP}) should also have moved the *neighbors* of A_{LAMP}. Since this does not happen, A_{LSP} cannot be close to the same points that A_{LAMP} was. A similar reasoning applies if we consider that A_{LSP} is correct – it then follows that A_{LAMP} cannot be correctly placed with respect to its neighbors.

Apart from this salient dark-colored bundle, we see many shorter and light-colored bundles. These show smaller-scale displacements between the two projections. For instance, we see how the red points at the right of the left group (B_{LAMP}) are moved to the left (B_{LSP}) of the same group. As these bundles fan out relatively little, do not have many crossings, and they are short, it means that B_{LSP} is almost a *translation* to the left of B_{LAMP}, so the two projections depict the same structure of the left group. Also, we do not see any bundle exiting this left group. This means that both LAMP and LSP keep all points in this group together. Finally, in the bottom-right group we see just a very few short light-colored bundles. Most points in this group do not have any bundles connected to them. This means that e^{disp} for these points is very small (yielding thus very short, nearly transparent, bundles). From this, we infer that LAMP and LSP produce very similar layouts for this group. If users are interested only to spot the most salient differences between two projections, and want to ignore such small-scale changes, this can be easily obtained by mapping e_i^{disp} to bundle-edge transparency.

4.8. *Usage scenario*

Considering that the user is offered quite a few different views to analyze projection errors, each with specific features and goals, the next question arises: How to put all these views together to form a coherent *usage scenario* for a common analysis task? Below we propose such a usage scenario. The view names herein refer to the respective techniques presented earlier in this section.

Step 1. Start with the *Aggregated Error* view. This shows an overview of the error at all points, without a distinction between false or missing neighbors. Next, check if (a) there are regions or groups with substantial errors or (b) the overall error is low. Case (b) indicates that the projection is quite good and that nothing else needs to be improved. In case (a), continue with steps 2, 3, and 4.

Step 2. The *Missing Neighbors Finder* view can be enabled and disabled freely over the *Aggregated Error* view to show the most important missing neighbors between all points. The user should notice now whether this view shows bundles having high error values (*i.e.* dark-colored). If so, there are important missing neighbors between the groups connected by such bundles. These groups must be further analyzed with the *Group Analysis Views*. If not, *i.e.* the bundles are colored (light) gray, this tells that the projection is good and, although there are missing neighbors, they are in a low error range and should not threaten the projection interpretation.

Step 3. Points, groups or regions found problematic in steps 1 and 2 are now analyzed in more detail using the *False Neighbors* and *Missing Neighbors* views. For groups detected in step 1 the most important thing is to find out exactly what kind of error is present: Are they (a) wrongly placed with respect to each other and other close points (false neighbors) or (b) in relation to far away points that should be closer (missing neighbors)? For groups detected in step 2, the error is already identified from the beginning: They have a high rate of missing neighbors. In this case, the question to be answered is: Which points are exactly the problematic ones inside the detected groups, or where exactly do the relations (bundle edges) with the highest errors start and end from? By using these two views, the user should be able to establish exactly which are the more problematic points (or groups), and what kind of error these have.

Step 4. Knowing now where exactly errors occur, we consider the next questions: (1) Are such errors really a problem? (2) Do they show unexpected results related to how the projection should work with the provided data? (3) Are the problematic points important for the analysis task at hand? If questions (1-3) all answer 'no', then we have a good projection for our data and analysis task, and our analysis stops. If any question (1-3) answers yes, then the user must improve the projection of problematic points, as follows. If the user is a projection designer testing the accuracy of a new method, (s)he should go back to the algorithm and use the new insight gotten from this analysis to improve that algorithm. If the user has no access to the projection implementation, the solution is to re-execute the analysis from step 1 with either (i) a new projection algorithm that might better fit the specific data and task; or (ii) a new set of parameters for the same algorithm. The new results can be compared with the old ones to determine if the errors have decreased or if the errors moved into a new region where they are not as important for the task at hand. For the second task, the *Projection Comparison View* can be used.

5. Applications

We now use our views to study several projections for several parameter settings – thus, to explore the space P that controls the creation of a DR projection. First, we present the datasets used (Sec. 5.1), the studied projection algorithms (Sec. 5.2), and their parameters (Sec. 5.3). Next, we use our views to explore the considered parameter settings (Secs. 5.4, 5.5).

5.1. Description of Datasets

Apart from the Segmentation dataset used so far, we consider the following datasets:

Freefoto: contains 3462 images grouped into 9 unbalanced classes [47]. For each image, we extract 130 BIC (border-interior pixel classification) features. Such features are widely used in image classification tasks [48].

Corel: composed of 1000 photographs that cover 10 specific subjects. Similarly to the Freefoto dataset, we extract for each image a vector of 150 SIFT descriptors [49].

News: contains 1771 RSS news feeds from BBC, CNN, Reuters and Associated Press, collected between June and July 2011. The 3731 dimensions were created by removing stopwords, employing stemming and using term-frequency-inverse-document-frequency counts. We manually classified the data points based on the perceived main topic of the news feed resulting in 23 labels. Given the imprecision of the manual classification and the restriction to have one topic per point, the labels are unbalanced for a number of points. Also, for other points (with different labels), we can still have a high similarity of content.

Sourceforge: This publicly available dataset contains 24 software metrics computed on 6773 open-source C++ software projects from the sourceforge.net website [50]. Metrics include classical objet-oriented quality indicators such as coupling, cohesion, inheritance depth, size, complexity, and comment density [51], averaged for all source code files within a project.

5.2. Description of Projections

We detail next the projection algorithms whose parameter spaces we will next study. We chose these particular algorithms based on their availability of documented parameters, scalability, genericity, presence in the literature, and last but not least availability of a good implementation.

LSP: The Least Squares Projection [4] uses a force-based scheme to first position a subset of the input points, called control points. The remaining points in the neighborhood of the control points are positioned using a local Laplace-like operator. Overall, LSP creates a large linear system that is strong in local feature definition. LSP is very precise in preserving neighborhoods from the nD space to the 2D space.

PLMP: The Part-Linear Multidimensional Projection (PLMP) [10] addresses computational scalability for large datasets by first constructing a linear mapping of the control points using the initially force-placed control points. Next, this linear mapping is used to place the remaining points, by a simple and fast matrix multiplication of the feature matrix with the linear mapping matrix.

LAMP: Aiming to allow more user control over the final layout, the Local Affine Multidimensional Projection (LAMP) [3] provides a user-controlled redefinition of the mapping matrix over a first mapping of control points. LAMP also works by defining control points, which are used to build a family of orthogonal affine mappings, one for each point to project. LAMP has restrictions regarding the number of dimensions against the number of points. Also, LAMP cannot directly work with distance relations, *i.e.*, it needs to access the nD point coordinates. However, LAMP is very fast, without compromising the precision reached, for instance, by LSP. Both LSP and LAMP can be controlled by a number of parameters, such as the control point set.

Pekalska: Another class of projection techniques works with optimization strategies. These are, in general, quite expensive computationally. To improve speed, Pekalska *et al.* [21] first embeds a subset of points in 2D by optimizing a stress function. Remaining points are placed using a global linear mapping, much like LAMP and LSP.

ISOMAP: The ISOMAP technique [14] is an extension of classical Multidimensional Scaling (MDS) that aims to capture nonlinear relationships in the dataset. ISOMAP replaces the input distance between point pairs by an approximation of the geodesic distance given by the shortest path on a graph created connecting neighbor points in the original space with the original distance as weight. The final 2D coordinates are computed via a conventional MDS embedding with calculations of eigenvalues over the distance relations of the previous step.

5.3. Description of parameters to analyze

Most techniques that initially project control points use a simplified iterative force-based algorithm, such as the one of Tejada *et al.* [27]. The number of *iterations* of force-based placement influences the control points' positions, and is, thus, a relevant parameter. LSP control points are typically the centroids of clusters obtained from a clustering of the input dataset. The *number of control points* is thus a second relevant parameter for LSP. To position points in the neighborhood of a given control point, LSP solves a linear system for that neighborhood. The neighborhood size (*number of neighbors*) is a third relevant parameter.

In LAMP, the affine mappings are built from a neighborhood of control points. The size of the control point set used to build the mapping, expressed as a *percentage* of the size of the control point set, is the main parameter here. The choice of control points and the choice of the initial projection of the control points are also parameterizable, just as for LSP, PLMP, and Pekalska. However, in LAMP, these parameters are mainly interactively controlled by the user, and thus of a lesser interest to our analysis.

ISOMAP, just as the previous methods, also requires the expression of neighborhoods. The main, and frequently only, exposed parameter of ISOMAP is the number of *nearest neighbors* that defines a neighborhood.

5.4. Overview comparison of algorithms

To form an impression about how the goals outlined in Sec. 3 are better, or less well, satisfied by LAMP, LSP, PLMP, and Pekalska, we start with an overview comparison.

Figure 8 shows the false neighbors, aggregated error, and most important $\phi = 5\%$ missing neighbors for the Segmentation dataset. To ease comparison, color mapping is normalized so that the same colors indicate the same absolute values in corresponding views. The aggregate error (top row) is quite similar in both absolute values and spread for all projections, *i.e.*, lower at the plot borders and higher inside, with a few dark (maximum) islands indicating the worse-placed points. Overall, thus, all studied projections are quite similar in terms of distance preservation quality. The false neighbors views (middle row) show a similar insight: Border points have few false neighbors (light colors), and the density of false neighbors increases gradually towards the projections' centers. Although local variations exist, these are quite small, meaning that all studied projections are equally good from the perspective of (not) creating false neighbors. The missing neighbors view (bottom row) is however quite different: By looking at the size and color of the depicted bundles, we see that LSP and Pekalska have much more important missing neighbors than PLMP, while LAMP has the fewest missing neighbors. In all cases, we see bundles that connect borders of the projected point-set. This confirms that all studied projections optimize placement of close points than far-away points. We also see that the missing neighbors are spread differently over the data: For LAMP, there are no bundles going to the bottom-right point cluster, showing that this cluster is indeed well separated in the projection, as it should be in relation to the *n*D data. In contrast, LSP, PLMP, and Pekalska all have bundles going to this cluster, indicating that they place these points too close to the remaining projected points.

5.5. Parameter analysis

We next refine our overview analysis by selecting two of the studied algorithms: LAMP and LSP. We next vary several of their parameters, and evaluate the resulting projections' quality with respect to this variation.

LAMP - Different control point percentages: Fig. 9 shows the results of LAMP for the Freefoto dataset with three different values for the *percentage* parameter: 10%, 30% and 50%. The error has been normalized on each view type (column in the figure).

First, we see that the final layout of the point cloud does not change drastically while varying the *percentage* parameter, only showing a 90 degree clockwise rotation for the value of 30%. While analyzing the false neighbors view, we also see that, while the light brown areas are large – meaning that a moderate amount of error can be expected on the whole layout – the dark-colored spots are found nearer to the center. This suggests that LAMP positions the most problematic points in the center, surrounded by the rest of the points. By focusing on the dark spots (points with the largest false neighbor errors) throughout the parameter variation we can see that the value of the largest errors on each result remain similar – no view has many more, or much darker-colored, areas.

For the missing neighbors view, we selected a point near the upper border of the layout, marked by a cross in Figs. 9 (b), (e) and (h)), since missing neighbors occur mainly on the borders of the projection, as we have already observed in Section. 4.4. The dark spot in Fig. 9 (h) is where the largest error occurs over these three views. While in Fig. 9 (b) there are a few orange spots showing moderate error, in Fig. 9 (e) the error decreases considerably, and then increases again in Fig. 9 (h). This suggests that using about 30% of neighbors is a good value for avoiding large numbers of missing neighbors. We confirmed this hypothesis on several other datasets (not shown here for brevity). Finally, the aggregated error view shows results very similar to the false neighbors view: More problematic points (dark spots) are pushed to the center, and moderate error is found spread evenly over the entire layout. This shows that, for LAMP, most errors come from false neighbors rather than from missing neighbors.

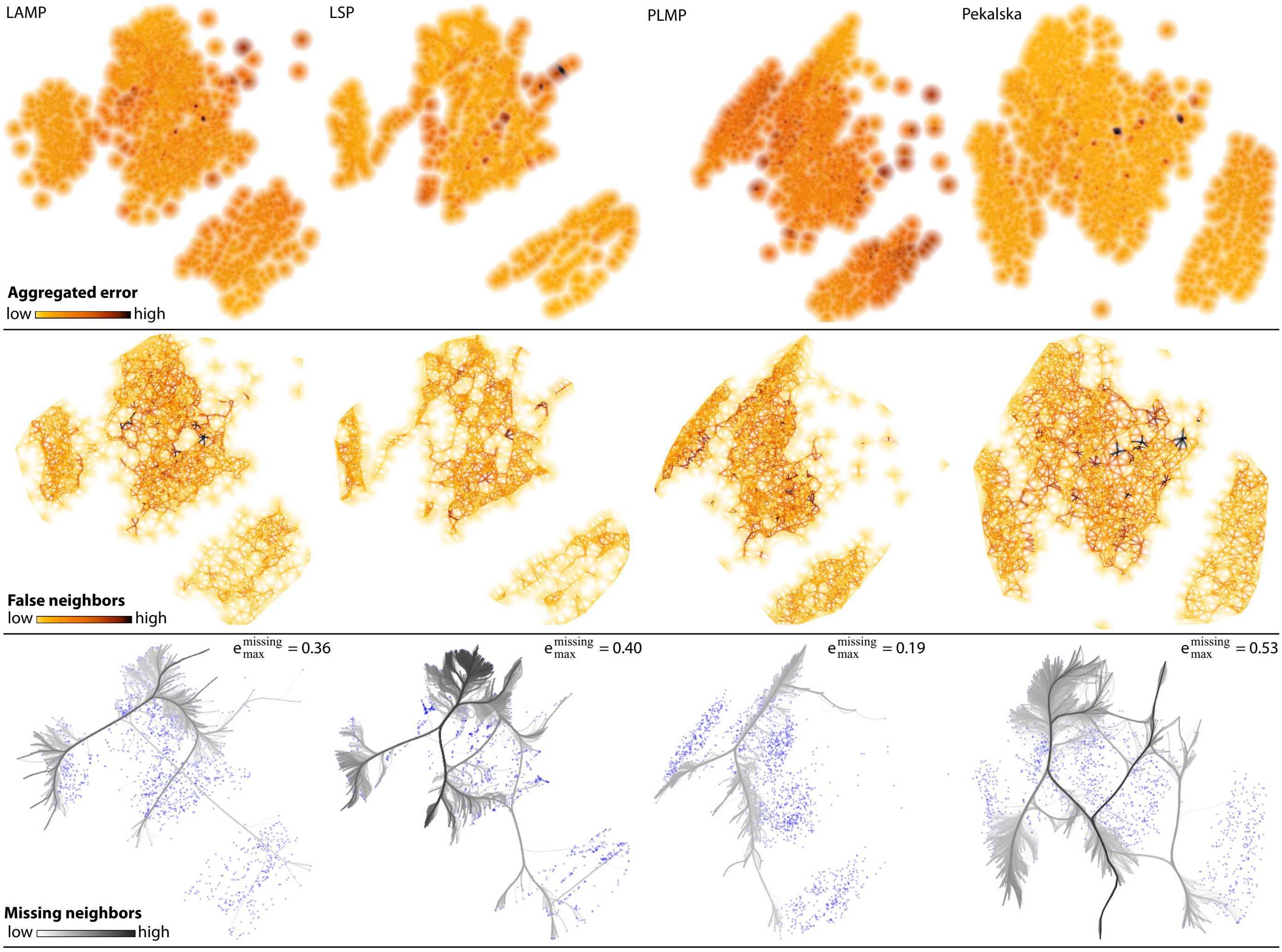

Figure 8: Comparison of LAMP, LSP, PLMP, and Pekalska projections for the Segmentation dataset (see Sec. 5.4)

LSP - Different numbers of control points: Figure 10 shows the same dataset (Freefoto) projected with LSP. The varying parameter is the *number of control points*. We use here the same views as in Fig. 9, and normalized the error in each column. By looking at the false neighbors views, we see a spatial interleaving of light-yellow and orange-brown colored areas in the projection. This contrasts with LAMP (Fig. 9) where the larger missing neighbor errors are consistently located away from the projection border. As the *number of control points* increases, the large error areas get more compact and closer to the projection center, but we see no increase in error severity (the amount of the orange and dark-red spots stays the same). In the missing neighbors views, the dark-colored areas in Fig. 10 (b) disappear largely in images (e) and (h), which means that the missing neighbors severity decreases when our control parameter increases. Comparing this with LAMP (Fig. 9 b,e,h), this shows that LAMP and LSP behave in opposite ways when dealing with missing neighbors. Finally, like for LAMP, the aggregate error views show the worst errors (dark spots) located in the center: The most problematic points are pushed inside by the other points which surround them, creating a mix of both false neighbors and missing neighbors. The severity of the errors, however, does not change visibly between the three parameter values.

LSP - Different numbers of neighbors: We next examine the effect of a second parameter of LSP: *number of neighbors*. For the Freefoto dataset, we fix 250 control points and vary the number of neighbors to 10, 50 and 100. Fig. 11 shows the results with the missing neighbors finder view. We see that the most significant errors are initially concentrated between groups A, B and C, with C being essentially too far placed from both A and B. Increasing our parameter reduces has a positive impact on solving the missing neighbors problem between groups A and C, bringing them together into the group marked AC. The main missing neighbors are now concentrated in the relationship between groups AC and B. The ‘concentration’ of error given by the parameter increase is, upon further analysis, explainable by the working of LSP: Given a neighborhood N, LSP’s Laplace technique positions all points in N close to each other in the final layout. However, the position of the neighborhoods N_i *themselves* is given only by the control

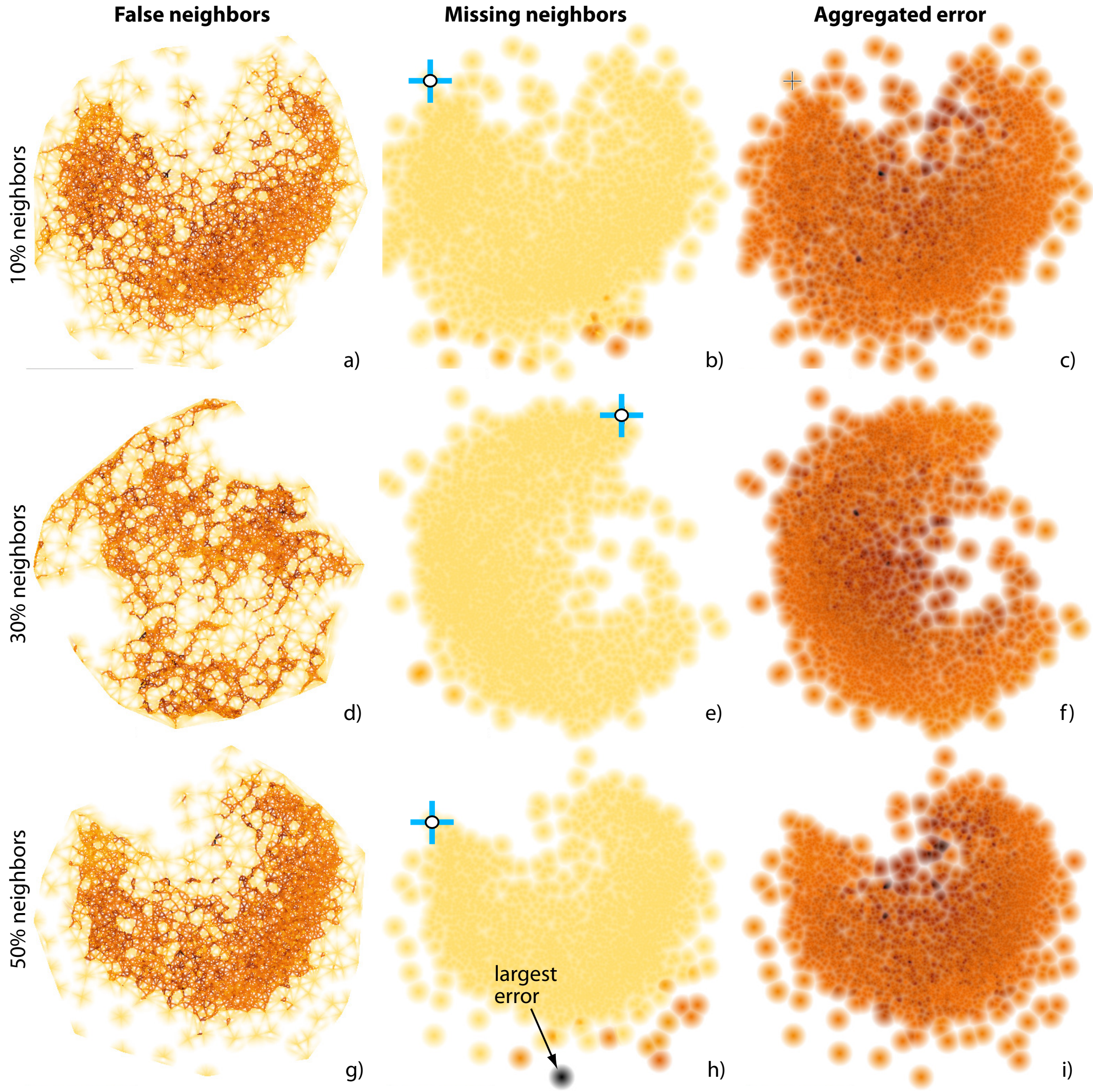

Figure 9: Applications – LAMP algorithm, Freefoto dataset, different neighbor *percentages* per row (see also Fig. 10).

points, which are determined by the initial force-based layout. If this layout suboptimally places two control points i and j too far away from each other, then *all* points within the neighborhoods N_i and N_j end up being too far away from each other. Hence, as the neighborhood size increases, the likelihood to see fewer thick high-error bundles increases. This insight we found is interesting since it was not reported in the LSP literature so far, and it can be explained (once we are aware of it) by the algorithmics of LSP.

LAMP - Different datasets: We next analyze the LAMP technique applied to three different datasets: Corel (1000 elements), Freefoto (3462 elements), and Sourceforge (6773 elements). The varying parameter is now the input *dataset* itself. The aim is to see whether (and how) errors are affected by the nature of the input data, *e.g.* distribution of similarity, number of dimensions, and number of points. Figure 12 top row shows the false neighbors views. We see here that, while for the first two datasets the behavior of false neighbors is similar to earlier results, for the largest dataset (Sourceforge) there are much fewer false neighbors. These are located close to the intersection area of the two apparent groups in the image, and on the borders of these groups. This, and the low errors (light colors) inside the groups may indicate that both groups have a high degree of cohesion between their inner elements. The large errors on close to the intersection areas and borders can indicate elements that could be in either group, respectively very different from all other elements. Figure 12 (a) shows a similar pattern: Most false neighbors are located at the ‘star’ shape’s center, while the arms of the start contain elements that are more cohesive. This may indicate that the dataset contains a number of cohesive groups equal to the number of start arms, and elements in the center belong equally to all groups.

While analyzing the missing neighbors for several points selected on the periphery of the projections, we see that the errors are smaller for Figs. 12 (d) and (e), and considerably larger for Fig. 12 (f). For the last image, we selected a point close to the intersection area of the perceived groups. Image (f) shows that this point is *equally* too far placed from most points

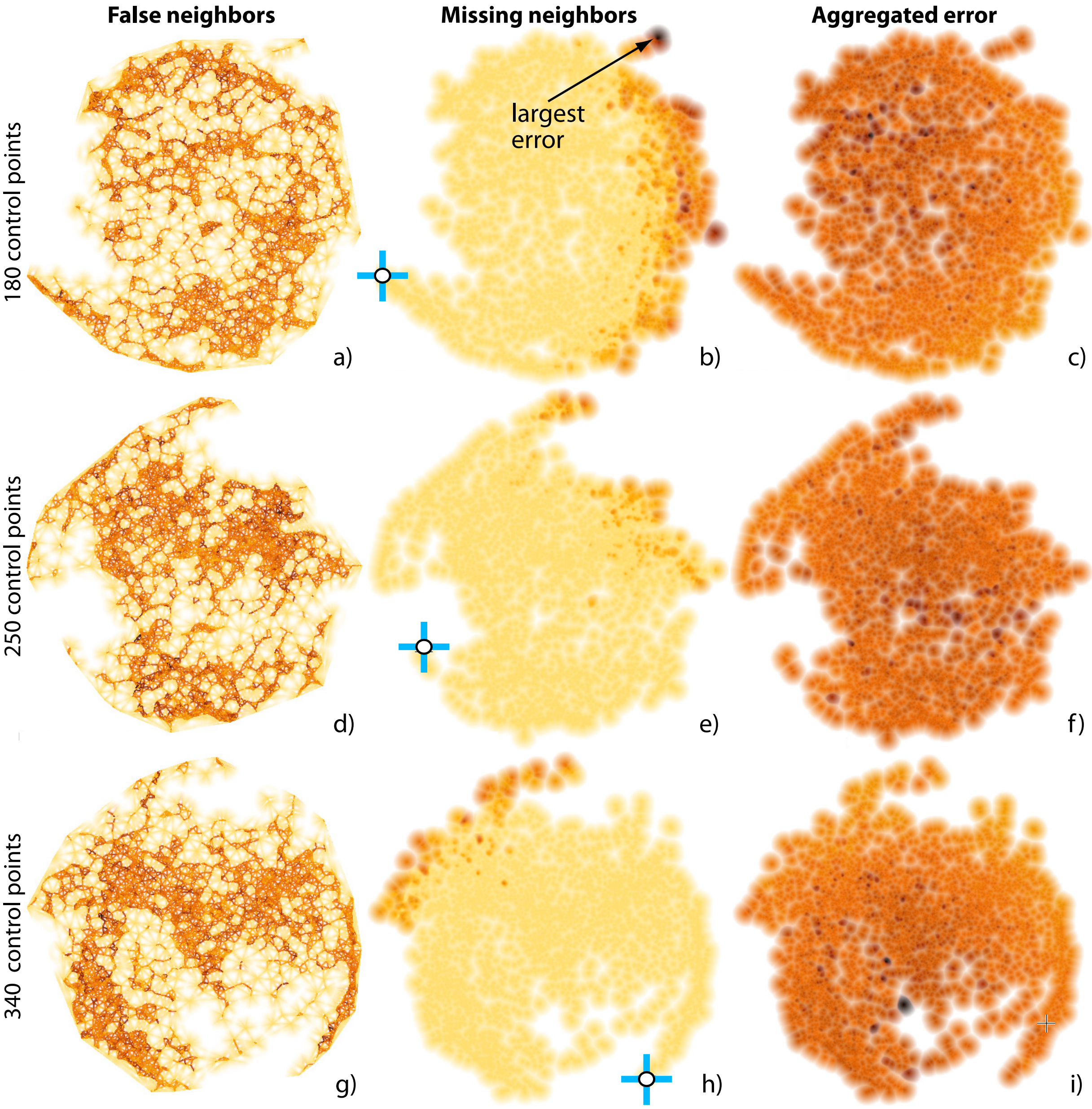

Figure 10: Applications – LSP technique, Freephoto dataset, different numbers of *control points* per row (compare with Fig. 9)

in *both* perceived clusters. The size and speed of increase of the error (as we get further from this point in the projection space) strongly suggests that the selected point belongs stronger to both perceived groups than the projection indicates. This strengthens our initial hypothesis that the area separating the two groups belongs equally to these groups.

ISOMAP - Different numbers of neighbors: To illustrate a different type of analysis made possible by our work, Fig. 13 shows the effect of changing the number of *neighbors* in ISOMAP on missing group members. Our group Γ of interest, shown first on Fig. 13 (a), is highlighted in images (b-d) by a shaded cushion. Besides the fact that Γ moves from the left of the projection to the right, images (b-d) show how its missing members behave as we change our parameter. At first, in Fig. 13 (b), we see that the most important missing neighbors are found in two other areas A_1 and A_2 on the far side of the layout. We also notice many black edges, which means that the points in A_1 and A_2 are indeed too far away from all points in the selected group. The relatively large fan-out of the bundles show that the group misses many members, and these are scattered widely over the projection. As the parameter increases, we see in image (c) that the missing members spread out even more, but the severity of the errors decreases (as shown by the lighter colors of $e_\Gamma^{missing}$ background. The inner fanning of the edges, inside Γ, is still large, which shows that many group members miss neighbors. Finally, in Fig. 13 (d), issues decrease significantly: We see thinner bundles, which imply less error; the bundle fanning inside Γ is relatively small, meaning that most of Γ's points do not miss neighbors; and the fan-out of the bundles is smaller, showing that the missing group members are now more concentrated than for the first

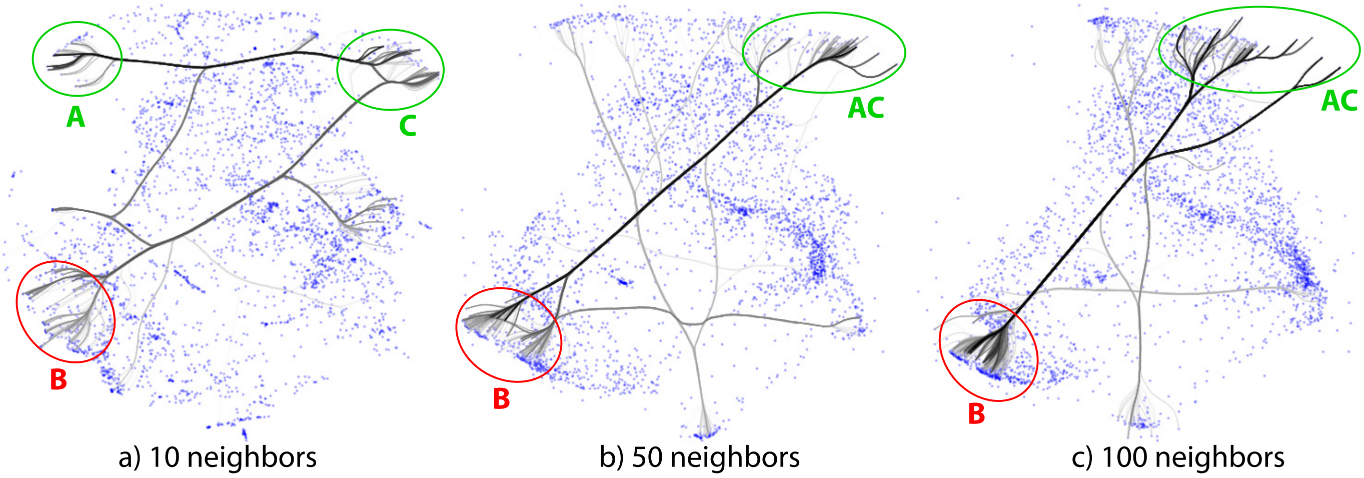

Figure 11: Applications – LSP technique, Freefoto dataset, different numbers of *neighbors*. Bundles show most important missing neighbors.

two parameter values. This leads to the conclusion that, for the analyzed group, the increase of the number of neighbors parameter has a positive impact on the final projection quality.

LSP - Different numbers of iterations: The final analysis we present compares two different LSP projections of the same dataset (News), computed using values of 50, respectively 100 for the *number of iterations* parameter of the control-point force-directed placement.

Figures 14 (a) and (b) show the two LSP projections. In each of them, several high-density groups are visible. These are strongly related news feeds, *i.e.*, which likely share the same topic (see Sec. 5.1). However, without extra help, we cannot *relate* the two projections, *e.g.*, find out (a) if points significantly change places due to the parameter change; (b) which groups in one projection map to groups in the other projection; and (c) whether points in a group in one projection are also grouped in the second projection.

To answer question (a), we use the projection comparison view (Sec. 4.7). The result (Fig. 14 (c)) shows that there are many large point shifts; the bundle criss-crossing also shows that groups change places in the projection. This is a first indication that LSP is not visually stable with respect to its number of iterations parameter. Next, we manually select three of the most apparent point groups in one projection, shown in Fig. 14 (a) by the shaded cushions A,B,C. We examine these in turn. In Fig. 14 (d), we show how points in group A shifted, in the second projection, to a group A_1. Virtually all bundled edges exiting A end in A_1, so the parameter change preserves the cohesion of group A (though, not its position in the layout). The same occurs for group B (Fig. 14 (e)). However, the parameter change spreads B more than A – in image (e), we see that B maps to three groups, $B_1..B_3$. These visualizations thus answer question (b). Group C behaves differently (Fig. 14 (f)): This group is split into two smaller groups C_1 and C_2 when we change our parameter. For question (c), thus, the answer is partially negative: not all groups are preserved in terms of spatial coherence upon parameter change.

6. Discussion

We have implemented our visualization techniques in C++ using OpenGL 1.1, and tested them on Linux, Windows, and OSX. Below we discuss several aspects of our method.

Computational scalability: For Delaunay triangulation and nearest-neighbor searches, we use the Triangle [52] and ANN [53] libraries. Both can handle over 100K points in subsecond time on a commodity PC. Further, we accelerate imaging operations using GPU techniques. For distance transforms, we use [54]. On an Nvidia GT 330M, this allows us to compute shaded cushions and perform our Shepard interpolation at interactive frame rates for views of 1024^2 pixels. For edge bundling, we implemented KDEEB [41] fully on Nvidia's CUDA platform. This yields a speed-up of over 30 times (on average) as compared to the C# implementation in [41] and allows bundling graphs of tens of thousands of edges in roughly one second. All in all, we achieve interactive querying and rendering of our views for projections up to 10K points.

Visual scalability: Our image-based approaches scale well to thousands of data points or more, even when little screen space is available. Moreover, all our techniques have a multiscale aspect: The parameters α and β (Eqns. 6, 7) effectively control the visual *scale* at which we want to see false neighbors, missing neighbors, and the aggregate error. Increasing these values eliminates spatial outliers smaller than a given size, thereby emphasizing only coarse-scale patterns (see *e.g.* Fig. 1). The bundled views (Sec. 4.5) also naturally scales to large datasets given the inherent property of bundled edge layouts to emphasize coarse-scale connectivity patterns.

Genericity: Our visualizations are applicable to any DR algorithm, as long as one can compute an error distance matrix encoding how much 2D distances deviate from their nD counterparts (Eqn. 2). No internal knowledge of, or access to, the DR algorithms is needed – these can be employed as black boxes. This allows us to easily compare widely different DR

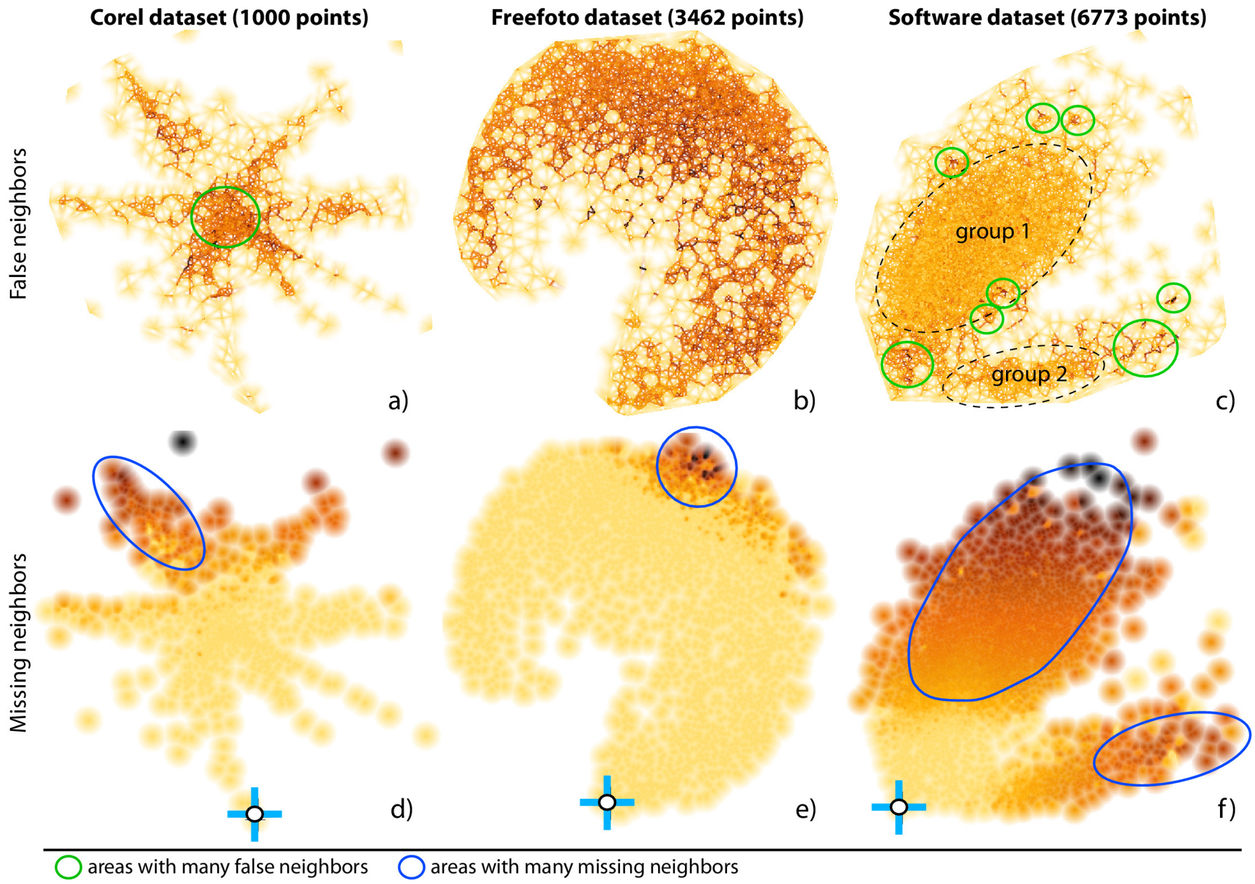

Figure 12: Applications – One algorithm (LAMP), different datasets. Top row: false neighbors. Bottom row: missing neighbors.

algorithms, *e.g.* based on representatives, based on distance matrices, or based on direct use of the *n*D coordinates.

Ease of use: Our views are controlled by three parameters: α sets the scale of the visual outliers we want to show; β sets the radius around a point in which we want to display information, *i.e.*, controls the degree of space-filling of the resulting images; ϕ sets the percentage of most important missing neighbors we want to show. These parameters, as well as the interaction for selecting point groups (Sec. 4.6) are freely controllable by users by means of sliders and point-and-click operations.

Comparison: Similarly to Van der Maaten *et al.* [32], we use multiple views showing the same data points to explain a projection, *e.g.*, the false neighbors, missing neighbors view, missing neighbors finder, and group-related maps. However, the multiple maps in [32] are used to actually convey the projection, so the same point can have different locations and/or weights in different maps. In contrast, we use multiple views to convey different quality metrics atop of the same 2D projection. Similar to Aupetit [33], our error metrics encode discrepancies in distances in $\mathbb{R}^n$ *vs* $\mathbb{R}^2$. However, our error metrics are different. More importantly, our visualizations are different: Our false neighbors view does not show (a) spurious Voronoi cell edges far away from data points or (b) cell subdivision edges whose locations does not convey any information, since we (a) use distance-based blending and (b) continuous rather than constant per-cell interpolation (Sec. 4.3). Secondly, our missing neighbors finder (Sec. 4.5) can show one-to-many and many-to-many error relationships, whereas all other methods are constrained to one-to-one relationships. Finally, we can show errors at group level, whereas the other studied techniques confine themselves to showing errors at point level only.

Our projection comparison view is technically related to the method of Turkay *et al.*, which connects two 2D scatterplots to each other by lines linking their corresponding points [55]. However, Turkay *et al.* stress that line correspondences only work for a *small* number of points. In contrast, we use bundles to (a) show up to thousands of correspondences, and coloring and blending to encode correspondence importance.

Findings: It can be argued that our results are limited, as we did not decide, using our method, which of the studied DR algorithms are best. However, this was not the aim of our work. Rather, our goal was to present a set of visual techniques that help analyze the effect of parameters on projection quality for several DR techniques of interest. Deciding whether a certain degree of quality, *e.g.* in terms of false neighbors, missing neighbors, grouping problems, or projection stability is a highly context, dataset, and application-dependent task. Having such a context, our tools can be then used to assess (a) which are the quality problems, (b) how parameter settings affect them, and (c) whether these problems are acceptable for the task at hand. The same observation applies to the

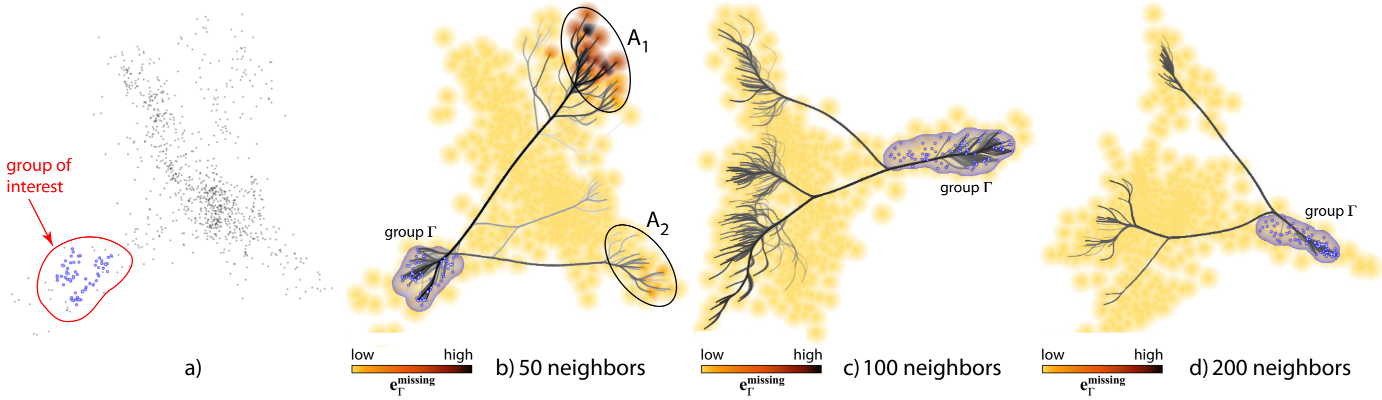

Figure 13: Applications – ISOMAP projection, finding missing group members for different numbers of *neighbors*.

datasets used here. Our analyzes involving these should be seen purely as test cases for assessing the quality problems of DR projections, and not as findings that affect the underlying problems captured by these datasets.

Limitations: As outlined by our examples, our visualizations can show (a) which projection areas suffer from low quality; and (b) how two projections differ in terms of neighborhood preservation. However, we cannot directly explain (c) *why* a certain DR algorithm decided to place a certain point in some position; and (d) how the user should *tune* (if possible) the algorithm's parameters to avoid errors in a given area. In other words, we can explain the function $f : P$ (Eqn. 1) and its first derivatives over P, but not the inverse f^{-1}. This is a much more challenging task – currently not solved by any technique we know of. Further explaining such second-order effects to help users locally fine-tune a projection is subject to future work. Secondly, the parameter space P of some DR algorithms can be high-dimensional. So far, we can only analyze the variation of one or two parameters at a time. Extending this to several parameters is a second challenging next topic.

7. Conclusions

We have presented a set of visualization methods for the analysis of quality of dimensionality-reduction (DR) algorithms by exploration of their parameter settings. By generically modeling such algorithms as functions from nD to 2D in terms of their distance-preservation error, we propose several views for assessing the distribution of false neighbors, missing neighbors, and aggregated projection error at both individual point and point-group level. We use several dense-pixel, visually scalable, techniques such as multi-scale scattered point interpolation and bundled edges to make our methods visually and computationally scalable to large datasets and also work in a multiscale mode. We demonstrate our techniques by analyzing the parameters of five state-of-the-art DR techniques. In contrast to existing assessments of DR projections by aggregate figures, that can only infer overall precision, we offer more local tools to examine how neighborhoods and groups are mapped in the final projection. The usage of our techniques is simple and, most importantly, allows users of DR techniques to study their quality without needing to understand complex internal processes or the exact role of each parameter in the projections.

Future work can target several directions. First, we plan to support 'what if' scenarios, *i.e.*, help users to decide how they could correct local projection problems by shifting wrongly-placed points while dynamically assessing the ensuing overall projection errors. Secondly, we plan to explicitly visualize the *reasons* that determine point placement, *i.e.*, depict the nD variable values which cause points to be placed close to, or far away from, each other. Additionally, we intend to provide tools for local evaluation of projections customized for specific target audiences. By this, we hope to make the operation of DR algorithms more transparent and understandable for users ranging from algorithm designers to end-users.

Acknowledgements

This research was financially supported by the grant PN-II-RU-TE-2011-3-2049 offered by ANCS, Romania, and by the research project CAPES/NUFFIC 028/11. Authors also wish to acknowledge the Brazilian financial agencies CNPq and FAPESP for their support.

References

[1] Elmqvist N, Dragicevic P, Fekete JD. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. IEEE TVCG 2008;14:1141–8.

[2] Heinrich J, Weiskopf D. Continuous parallel coordinates. IEEE TVCG 2009;15(6):1531–8.

[3] Joia P, Paulovich FV, Coimbra D, Cuminato JA, Nonato LG. Local affine multidimensional projection. IEEE TVCG 2011;17:2563–71.

[4] Paulovich FV, Nonato LG, Minghim R, Levkowitz H. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. IEEE TVCG 2008;14(3):564–75.

[5] Cui W, Wu Y, Liu S, Wei F, Zhou MX, Qu H. Context-preserving, dynamic word cloud visualization. IEEE CG&A 2010;:42–53.

[6] Faloutsos C, Lin KI. FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: Proc. ACM SIGMOD Intl. Conf. on Management of Data. 1995, p. 163–74.

[7] Chen Y, Wang L, Dong M, Hua J. Exemplar-based visualization of large document corpus. IEEE TVCG 2009;15:1161–8.

[8] Paulovich FV, Nonato LG, Minghim R. Visual mapping of text collections through a fast high precision projection technique. In: Proc. IEEE Information Visualization. 2006, p. 282–90.

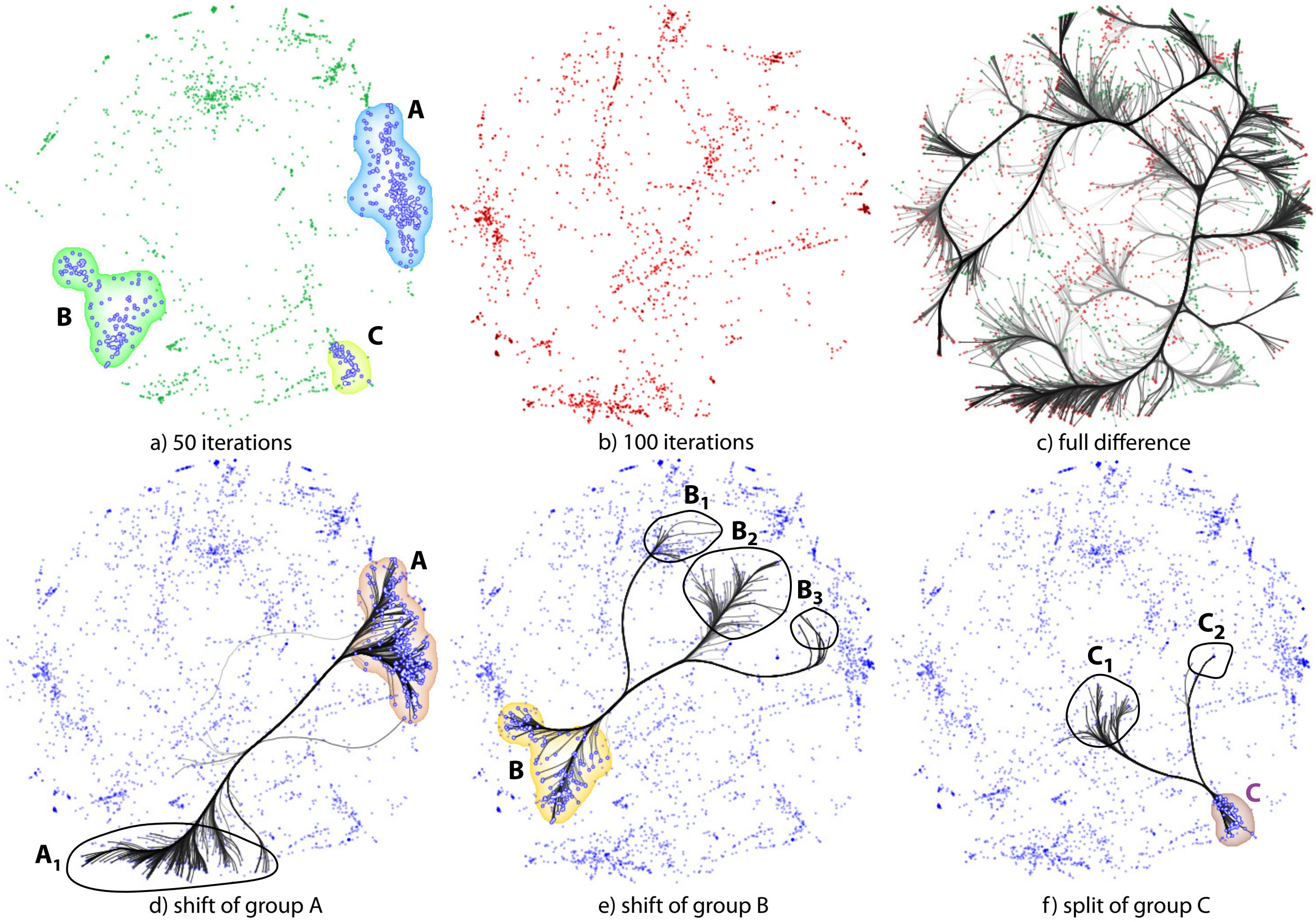

Figure 14: Applications – Shift between two LSP projections, for different numbers of force-directed *iterations*.

[9] Daniels J, Anderson EW, Nonato LG, Silva CT. Interactive vector field feature identification. IEEE TVCG 2010;16:1560–8.

[10] Paulovich FV, Silva C, Nonato LG. Two-phase mapping for projecting massive data sets. IEEE TVCG 2010;16:1281–90.

[11] Poco J, Eler D, Paulovich FV, Minghim R. Employing 2D projections for fast visual exploration of large fiber tracking data. CGF 2012;31:1075–84.

[12] Torgeson WS. Multidimensional scaling of similarity. Psychometrika 1965;30:379–93.

[13] Roweis ST, Saul LK. Nonlinear dimensionality reduction by locally linear embedding. Science 2000;290(5500):2323–6.

[14] Tenenbaum JB, de Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science 2000;290(5500):2319–23.

[15] Silva VD, Tenenbaum JB. Global versus local methods in nonlinear dimensionality reduction. In: Advances in Neural Information Processing Systems; vol. 15. MIT Press; 2003, p. 705–12.

[16] de Silva V, Tenenbaum J. Sparse multidimensional scaling using landmark points. Tech. Rep.; Stanford Univ.; 2004. `window.stanford.edu/courses/cs468-05-winter/Papers/Landmarks/Silva_landmarks5.pdf`. Last access: 15/07/2013.

[17] Brandes U, Pich C. Eigensolver methods for progressive multidimensional scaling of large data. In: Proc. Graph Drawing. Springer; 2007, p. 42–53.

[18] Gansner ER, Koren Y, North S. Graph drawing by stress majorization. In: Proc. Graph Drawing. Springer; 2004, p. 239–50.

[19] Sammon JW. A nonlinear mapping for data structure analysis. ACM Trans Comp 1969;C-18:401–9.

[20] Bronstein M, Bronstein A, Kimmel R, Yavneh I. Multigrid multidimensional scaling. Numer Linear Algebr 2006;13:149–71.

[21] Pekalska E, de Ridder D, Duin R, Kraaijveld M. A new method of generalizing Sammon mapping with application to algorithm speed-up. In: Proc. 5^{th} Annual Conference of the Advanced School for Computing and Imaging (ASCI). Delft, NL; 1999, p. 221–8.

[22] Eades PA. A heuristic for graph drawing. Congressus Numerantium 1984;42:149–60.

[23] Chalmers M. A linear iteration time layout algorithm for visualising high-dimensional data. In: Proc. IEEE Visualization. 1996, p. 127–31.

[24] Frishman Y, Tal A. Multilevel graph layout on the gpu. IEEE TVCG 2007;13:1310–9.

[25] Ingram S, Munzner T, Olano M. Glimmer: Multilevel MDS on the GPU. IEEE TVCG 2009;15(2):249–61.

[26] Jourdan F, Melangon G. Multiscale hybrid MDS. In: Proc. IEEE Information Visualization. 2004, p. 388–93.

[27] Tejada E, Minghim R, Nonato LG. On improved projection techniques to support visual exploration of multidimensional data sets. Inf Vis 2003;2(4):218–31.

[28] Geng X, Zhan DC, Zhou ZH. Supervised nonlinear dimensionality reduction for visualization and classification. IEEE Trans Syst Man Cybern 2005;35(6):1098–107.

[29] Sedlmair M, Tatu A, Munzner T, Tory M. A taxonomy of visual cluster separation factors. CGF 2012;31(3):1335–44.

[30] Schreck T, von Landesberger T, Bremm S. Techniques for precision-based visual analysis of projected data. Inf Vis 2010;9(3):181–93.

[31] van der Maaten L, Hinton G. Visualizing data using t-SNE. J Mach Learn Res 2008;9:2431–56.

[32] van der Maaten L, Hinton G. Visualizing non-metric similarities in multiple maps. Machine Learning 2012;87(1):33–5.

[33] Aupetit M. Visualizing distortions and recovering topology in continuous projection techniques. Neurocomputing 2007;10(7-9):1304–30.
[34] Lespinats S, Aupetit M. CheckViz: Sanity check and topological clues for linear and non-linear mappings. CGF 2011;30(1):113–25.
[35] Heulot N, Aupetit M, Fekete JD. Proxilens: Interactive exploration of high-dimensional data using projections. In: EuroVis Workshop on Visual Analytics using Multidimensional Projections. 2013, p. 11–5.
[36] Frank A, Asuncion A. UCI machine learning repository. 2013. `www.ics.uci.edu/~mlearn`. Last access: 15/07/2013.
[37] Paulovich F, Eler D, Poco J, Botha C, Minghim R, Nonato LG. Piece wise Laplacian-based projection for interactive data exploration and organization. CGF 2011;30(3):1091–100.
[38] Silva C, Paulovich F, Nonato LG. User-centered multidimensional projection techniques. Comput Sci Eng 2012;14(4):74–81.
[39] Brewer C, Harrower M. ColorBrewer. 2013. URL: `http://www.colorbrewer.org`.
[40] Rumpf M, Telea A. A continuous skeletonization method based on level sets. In: Proc. VisSym. Eurographics; 2002, p. 151–9.
[41] Hurter C, Ersoy O, Telea A. Graph bundling by kernel density estimation. CGF 2012;31(3):865–74.
[42] Comaniciu D, Meer P. Mean shift: A robust approach toward feature space analysis. IEEE TPAMI 2002;24(5):603–19.
[43] Ester M, Kriegel HP, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. 2^{nd} Intl. Conf. on Knowledge Discovery and Data Mining. 1996, p. 226–31.
[44] Jain AK, Murthy M, Flynn P. Data clustering: A review. ACM Comput Surv 1999;31(3):264–323.
[45] Jain AK. Data clustering: 50 years beyond k-means. Pattern Recognit Lett 2010;31(8):651–66.
[46] Telea A, Ersoy O. Image-based edge bundles: simplified visualization of large graphs. CGF 2010;29(3):843–52.
[47] Freefoto . Free stock photo collection. 2013. `www.freefoto.com`. Last access: 15/07/2013.
[48] Stehling RO, Nascimento MA, Falcão AX. A compact and efficient image retrieval approach based on border/interior pixel classification. In: Proc. CIKM. ACM; 2002, p. 102–9.
[49] Li J, Wang JZ. Automatic Linguistic Indexing of Pictures by a Statistical Modeling Approach. IEEE TPAMI 2003;25:1075–88.
[50] CCSL/IME-USP . The complete data set used within the paper "a study of the relationships between source code metrics and attractiveness in free software projects". 2013. `http://ccsl.ime.usp.br/mangue/data`. Last access: 15/07/2013.
[51] Lanza M, Marinescu R. Object-Oriented Metrics in Practice. New York: Springer; 2006.
[52] Shewchuk JR. Delaunay refinement algorithms for triangular mesh generation. Computational Geometry: Theory and Applications 2002;22(1–3):21–74.
[53] Mount D, Arya S. Approximate nearest neighbor search software. 2011. `www.cs.umd.edu/~mount/ANN`. Last access: 15/07/2013.
[54] Cao TT, Tang K, Mohamed A, Tan TS. Parallel banding algorithm to compute exact distance transform with the GPU. In: Proc. ACM I3D. 2010, p. 83–90.
[55] Turkay C, Filzmoser P, Hauser H. Brushing dimensions-a dual visual analysis model for high-dimensional data. IEEE TVCG 2011;17(12):2591–9.