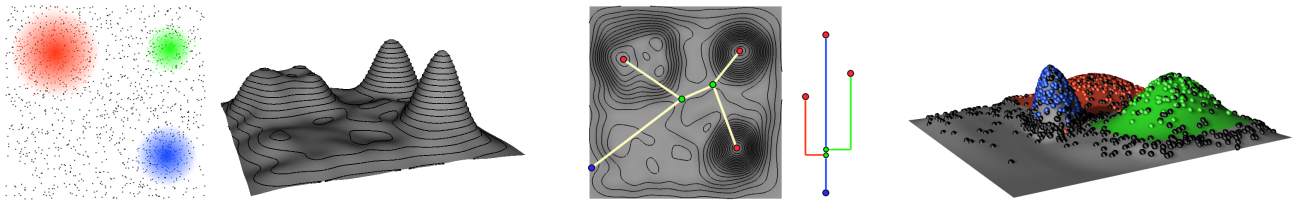
Visual Analysis of High Dimensional Point Clouds using Topological Landscapes

Patrick Oesterling* University of Leipzig
Christian Heine† University of Leipzig
Heike Jänicke‡ Swansea University
Gerik Scheuermann§ University of Leipzig



(a) visualizable only up to three dimensions (b) calculable in arbitrary dimensions (c) 3-D terrain

Figure 1: (a) Because intuitive spatial scatter plot visualizations are limited to, at most, three dimensions, we cannot *look* into a high dimensional data set by simply visualizing its point cloud or, e.g., its density distribution. (b) However, the topology of a point cloud, described, e.g., by means of the contour tree, can be calculated in arbitrary dimension. (c) Instead of the data itself, we visualize its topological landscape [36], having the same topology as the given input data set.

Abstract

In this paper, we present a novel three-stage process to visualize the structure of point clouds in arbitrary dimensions. To get insight into the structure and complexity of a data set, we would most preferably just *look* into it, e.g. by plotting its corresponding point cloud. Unfortunately, for orthogonal scatter plots, this only works up to three dimensions, and other visualizations, like parallel coordinates or scatterplot matrices, also have problems handling many dimensions and visual overlap of data entities.

The presented solution tackles the problem of visualizing point clouds indirectly by visualizing the topology of their density distribution. The benefit of this approach is that this topology can be computed in arbitrary dimensions. Similar to examining scatter plots, this gives the important information like the number, size and nesting structure of accumulated regions. We view our approach as an alternative to cluster visualization.

To create the visualization, we first estimate the density function using a novel high-dimensional interpolation scheme. Second, we compute that function's topology by means of the join tree, generate a corresponding 3-D terrain using the *topological landscape* metaphor introduced by Weber *et al.* [36], and finally augment that landscape by placing the original data points at suitable locations.

Index Terms: I.5.3 [Pattern Recognition]: Clustering—Algorithms; I.5.3 [Pattern Recognition]: Models—Structural; I.5.3 [Pattern Recognition]: Design Methodology—Pattern Analysis;

*e-mail: oesterling@informatik.uni-leipzig.de
†e-mail: heine@informatik.uni-leipzig.de
‡e-mail: h.jaenicke@swansea.ac.uk
§e-mail: scheuermann@informatik.uni-leipzig.de

IEEE Pacific Visualisation Symposium 2010
2 - 5 March, Taipei, Taiwan
978-1-4244-6686-3/10/$26.00 ©2010 IEEE

1 Introduction

Visualizing and studying multivariate data has a long tradition. For hundreds of years, and in almost every discipline, people used to organize data entities with several attributes in a table-like manner. For a small number of attributes, simple visualizations of data or statistical information can be achieved using scatter plots or histograms. For classified data, or when different data points show statistical similarity, these simple visualizations might be colored or handled with transparency in order to differentiate between several coherent parts of the data. For example, observing a scatter plot, parallel coordinate system or a dendrogram, leads to much more insight and understanding of the data structure, than staring at a spreadsheet or a collection of data tuples [34]. However, for some thousands of entities, having maybe some hundreds of attributes, most common visualizations fail at reasonably handling the dimensionality or the overlap-free data arrangement. With increasing size of the data, it also becomes increasingly complicated to bring the essential information to the point without forcing the user to consider a rather big part of the whole information in detail.

Hence, the goal is to break the habit of applying non-scalable approaches for small low-dimensional data sets to big high-dimensional data sets by using more intuitive metaphors which make complex information easier to understand. One such metaphor is the topological landscape, which takes advantage of the fact that humans are naturally trained in understanding the structure of a terrain. By means of a topological landscape, the data is visualized indirectly by only showing its topological information, instead of the data itself. There are several advantages: First of all, the topology can be calculated for point clouds in arbitrary dimensions. Secondly, while we cannot visualize the high dimensional data itself, we can (independent of the dimensionality of the underlying input space) map the topology to a 3-D terrain. In the end, by visualizing the topology, we directly give answers to structural questions, instead of presenting a visualization which has to be analyzed with other methods afterwards.

2 Related Work

Although we visualize the given point cloud indirectly, several direct methods are available: *Axis-based* approaches like parallel coordinates [21], star plot [10] or RadViz [5] try to overcome the three-dimensional limitation of common scatterplots [10] by using a non-orthogonal axis-alignment. Although this allows for a more effective study and visualization of correlations in high dimensional data, the order of the axes is crucial and difficult to determine automatically. Moreover, some of these methods need a huge amount of pixels to visualize single data entities and still suffer from pixel overlapping. For an efficient usage of screen pixels, *pixel-based* methods [24] were introduced to arrange the data in recursive patterns, thus unveiling trends in the data's attributes. *Projective* approaches like scatterplot matrix [6], Prosection View [17] or Hyperbox [3] are used to give an overview of all possible attribute configurations, i.e., they only consider two dimensions at a time. Consequently, the user only investigates a small subset and has to combine several different visualizations to get an adequate overview of the whole structure. In modern approaches like *Rolling the Dice* [14] the rather static analysis of scatterplot matrices is spiced up with flexible interactive methods, queries and visual expressiveness.

Because all the basic methods become problematic when the dimensionality of the data becomes significantly high, it is a common approach to reduce the dimensionality. This focuses on finding correlations in subspaces and reducing the computational complexity. *Transformation* methods like PCA [23], MDS [26] or Kohonen Maps [25] project the data onto a smaller space which aggregates a maximum of variance. *Feature selection* methods are used for data reduction by removing irrelevant or redundant dimensions. An extension to the latter approach is *subspace clustering*. Its idea is that different subspaces may contain different, meaningful clusters. This requires effective and efficient ways to determine these subspaces. By default, our approach considers *all* dimensions and therefore special structures in subspaces might not be found. In principle, we could also apply out method only to particular subspaces, thus accelerating our overall run-time and finding similar features.

To formally define clusters and their detection, several methods have been developed. *Density*-based methods, like DENCLUE [20] or DBSCAN [15], regard clusters as dense regions of objects that are separated by regions of low density (representing empty space or noise). *Grid*-based methods, like STING [35] or WaveCluster [31], use a multiresolution grid to quantize the object space into a finite number of cells that are used to perform the clustering operations. Especially for higher dimensions, some methods, like CLIQUE [2] and PROCLUS [1] were introduced to overcome matters like *curse of dimensionality* [7].

An approach closely related to ours was recently introduced by Takahashi *et al.* [32], who construct a 3-D visualization approximating a high-dimensional scalar function's topology. Their method is a variation of the Isomap approach [33], and requires that the scalar values are given for all points. Their method is also dependent on eight parameters and they provide empirically derived values for seven of them. For one parameter (k) they are vague and it is not clear how to select this parameter for irregular high-dimensional data. The highest dimension for which they demonstrate their method is 7. Our method includes the computation of the scalar (density) value and requires one parameter for that. Two more parameters are used to simplify the topological landscape [36], but the core of our algorithm is parameter free.

3 Background

3.1 Neighborhood Graphs

To extract a meaningful density function in a high dimensional space, we need methods to fill the void between data samples and methods to describe closeness between points. To that end, we present some structures that describe partitions of the space subject to closeness of points.

Let $\delta(x,y)$ denote the distance of two points x,y in a d-dimensional Euclidean space $\mathbb{R}^d$, the *Voronoi diagram* [16] of a point set $P = \{p_1, p_2, \ldots, p_n\}$ is a partition of $\mathbb{R}^d$ into cells $c_i = \{x \in \mathbb{R}^d | \forall j \neq i : \delta(x,p_i) \leq \delta(x,p_j)\}$. Each cell is a class of points having the same closest point of P.

A *simplicial complex* S is a topological space consisting of simplices (points, lines, triangles, tetrahedra, etc...) and satisfying that for each simplex, each of its faces is element of S and each intersection of a pair of simplices of S is their common face.

The *Delaunay triangulation* [16] of a point set P in $\mathbb{R}^d$ in so-called general position is a simplicial complex such that for each simplex there exists an empty circum-hypersphere. There is a well known duality: two cells c_i, c_j are neighbors in the Voronoi diagram if and only if the Delaunay triangulation contains a line between p_i and p_j. There are algorithms that can compute the Delaunay triangulation directly and they perform very well in two and three dimensions. For higher dimensions, it has been shown that the number of simplices is in $\Omega(n^{d/2})$ in the worst case [16], giving an exponential lower bound on the run-time. We will refer to the points and lines of the Delaunay triangulation, omitting all other simplices, as the *Delaunay graph*.

The *Gabriel graph* [18] of a point set P in $\mathbb{R}^d$ is a graph (P,E) which contains an edge between two points u,v if and only if the Gabriel lune contains no point from P except on its border. The *Gabriel lune* is the smallest hypersphere with both u and v on its border. The Gabriel graph can be computed using $O(n^3)$ distance calculations by the following simple algorithm: for each point pair $u,v \in P$ construct the center c of u and v and test each point $w \in P, w \neq u, w \neq v$ whether it is closer to c than half the distance of u and v. An edge is inserted only if no such point is found. The inner loop can be aborted as soon as the first point inside the lune is found. Note that a distance calculation in a high dimensional space usually takes $O(d)$ time, giving an overall run-time of $O(dn^3)$. Because each point pair can be tested independently, this algorithm is trivial to implement in parallel using up to $n(n-1)/2$ processors.

A *neighborhood graph* [22], also called a *proximity graph*, is a graph in which *similar* or *neighbored* points are connected with an edge. Many types of neighborhood graphs have been developed [4]. The Delaunay graph is a suitable neighborhood graph as it maps cell neighborhood relation to edges. In the Gabriel graph, two vertices are connected if some vicinity of their direct connection is devoid of points. Neighborhood graphs often are related. In fact, the Gabriel graph is a subgraph of the Delaunay graph. An edge (p_i, p_j) is Delaunay but not Gabriel if it does not intersect the boundary of the cells c_i, c_j of the dual Voronoi diagram. These are usually the longer edges of each simplex. In addition, the Gabriel is always connected, because the minimum spanning tree is a subgraph of it.

3.2 Topology

Data in scientific environments is often provided as a discrete scalar function (or scalar field), describing, e.g., pressure, density or temperature distributions. The task for the visualization is to illustrate the data's structure and complexity and to examine whether the data contains unexpected features which require a closer investigation.

For up to three dimensions, scalar data might be visualized directly by, e.g., volume rendering [27] or indirectly by, e.g., extracting isosurfaces [29]. However, these approaches always show only a subset of the data and require the user to combine information from different images. Furthermore, the resulting visualization heavily depends on parameter selection. Therefore, another approach to investigate the structure of a (primarily high dimensional) scalar function is to compute its topology. This aims at a complete study of the variation of the function, described by means of critical points or entire regions [37].

Let the scalar data be given as a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^d$, with corresponding scalar measurements $H = \{h_1, h_2, \ldots, h_n\}$. To interpolate values for points not in P, the data is extended to the entire space by means of a mesh, with vertex set P, and a continuous function f that satisfies $f(p_i) = h_i$. A *level set of f at some function value h* is the set $\{x \in \mathbb{R}^d \mid f(x) = h\}$ and may consist of zero, one, or more connected components, called *contours*. For lower dimensions, level sets are known as *isolines* (2-D) and *isosurfaces* (3-D). If the function value $f(x)$ is thought of as time, we can watch the evolution of contours of f over time, seeing them appear, join, split and disappear. The *contour tree* [8] is a graph in which each contour is contracted to a single point and which tracks these topological changes. In Figure 1(b), the height field illustrates the scalar function and the rings constitute some contours. Consequently, these rings start to appear on top of the hills, merge or split at several heights and finally disappear in sinks in the valley. The contour tree tracks these changes at their particular height values, thus representing the topology by means of critical points (maximum=appearance, saddle=join/split and minimum=disappearance of contours).

The computation of the contour tree usually requires a *Morse function*, i.e. a function that requires all critical points to occur on distinct function values. Unfortunately, our density function is, in general, not a Morse function, but we apply simulation of simplicity [13] to make it a Morse function.

If the mesh is a simplicial complex and piecewise linear interpolation is used, the contour tree can be determined by computing and merging two trees: the join tree and the split tree [9]. The join tree of a scalar function contains the global minimum and all local maxima as well as all saddles which join contours. Concerning the density function, we are only interested in areas of high density and their nesting and therefore the join tree is sufficient for our purposes. The construction of the join tree is a graph algorithm, i.e. only points and edges of the mesh are considered, but higher order simplices are ignored. For space reasons we refer to [9] where the reader can find a simple $O(n \log n)$ algorithm to compute the join tree (n and m denote the number of vertices and edges, respectively). However, we would like to remark that if the split tree is a single path of vertices, the contour tree is equal to the join tree. We can therefore always use an algorithm that is applicable to a contour tree on a join tree as well.

The contour tree is often partitioned into *branches* which are paths that are monotonic with respect to the Morse function values. Each branch's *persistence* [12] can be measured as the difference of its highest and its lowest function value. The branches can be organized in a *branch decomposition* [30] which is a tree. Each of this tree's nodes is a branch, the root being the branch of highest persistence. One node is child of another node if the branch it represents has lower persistence than the other node's branch and both branches are connected by a saddle. It is easy to see that in the case where the join tree equals the contour tree, the main branch always connects a global minimum with a global maximum because the path between these two extrema is always monotonic.

3.3 Topological Landscape

The topological landscape metaphor was introduced by Weber *et al.* [36]. Due to the page limitation, we cannot give an appropriate account of the algorithm to create the landscape because it would require us to introduce a number of concepts unrelated to the rest of our work. The generation of the topological landscape is fast and the landscapes are easily understood by humans. For our purposes, we will consider this stage to be a black box into which a branch decomposition is given and which gives back a list of triangles with endpoints in $\mathbb{R}^3$, where each triangle is annotated with the branch it originates from. This information is necessary to map the original data points to the correct region of the landscape. Figure 2 shows a contour tree and its corresponding landscape, having the same topology as the input data set. The mapping of branches and triangles is indicated by colors.

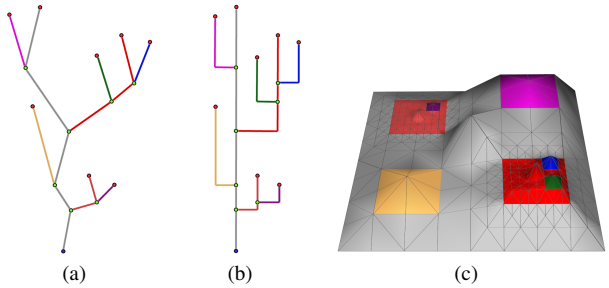


Figure 2: (a) A contour tree. (b) Its branch decomposition. (c) The topological landscape having the same contour tree. Colors have been used to mark which parts of the landscape correspond to which contour tree branches.

4 Algorithm

Given the high dimensional point cloud as a point set $P = \{p_1, p_2, \ldots, p_n\}$ in a fixed-dimensional Euclidean space $\mathbb{R}^d$ and a filter radius σ, we approximate the density function by sampling on this point set's Gabriel graph, compute that functions join tree and then generate the topological landscape from the join tree. Finally, we add the original data points as small spheres on the landscape to improve the perception of the density distribution on the hills. This augmentation has several applications:

1. We are able to study in what amount data points contribute to hills in the topological landscape given the current σ. This allows, among other things, to select an improved filter radius σ (see Section 4.4).

2. Assuming some pre-classified input, we can easily prove whether dense regions correspond to classes in the data by coloring the spheres accordingly, as we have done in all our examples. If class members are equal because they share similar values of several attributes, then the spheres with the same color should be placed solely on one hill in the topological landscape.

3. It also allows for augmenting the landscape, e.g. with labels, and for clicking on spheres to get more information. For a later interactive visualization, possibly incorporating semantic zoom, this can be of great benefit if the data points represent more than a high-dimensional vector.

4.1 Approximating the Density Function

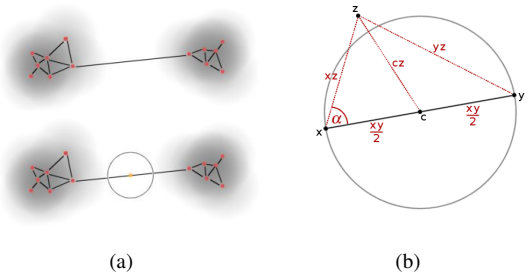
To compute the topology of the density function, we first need to generate it from the point set. Conceptually, we blur a high dimensional image that contains just the points using a *Gaussian filter*. Let $\delta(x, p_i)$ denote the Euclidean distance of any point x to a point p_i, the resulting image is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \sum_{i=1}^{n} exp\left(-\frac{\delta(x,p_i)^2}{2\sigma^2}\right) \quad (1)$$

We can omit the factor before the summation sign because it does not affect the nesting relations between dense regions.

Because the computation of this function's topology is very hard, we approximate f by a simpler function f' which has very similar topology and for which we can use the join tree computation. Our

aim is to make an f' that can be described by piecewise linear interpolation on a simplicial complex S. One option for S would be a Delaunay triangulation on P with $f'(p_i) = f(p_i), \forall p_i \in P$, but we cannot use that for two reasons: (i) The linear interpolation on S is a bad approximation for f as it does not reflect that the density between two points can be lower than the density at the two points, as illustrated in Figure 3(a). If so, f' lacks a corresponding saddle value which attests points to be separated by a low-density region. (ii) The run-time increases exponentially with the dimension d.



(a) (b)

Figure 3: (a) Without an additional low density sample in between two distant points, two dense regions would be identified as one by the join tree computation. The missing saddles are added by sampling the density function not only on the original points but also on the mid-points of edges. (b) Because the triangles xyz and xcz share the angle α, the distance of the mid-point c to point z can be computed directly from the distances between x, y, z using the law of cosines.

The first problem could be solved exactly by computing the Delaunay triangulation S' on P augmented by all critical points of f. But as we already noted, these critical points are difficult to compute. Especially in higher dimensions, most simplices contain one minimum, resulting in an exponentially growing number of minima with growing dimension. Furthermore, we are only interested in maxima and the saddles between them. Maxima usually lie near the points of S and the saddles lie near the midpoints of the edges of S. We therefore could compute the Delaunay triangulation on P and split each edge at its midpoint adding the additional points $v_1, v_2, \ldots, v_m$ for which we also request that $f'(v_i) = f(v_i)$. This means an additional m (= the number of edges of S) evaluations of f, each requiring an additional n distance calculations. Let c be the midpoint of an edge $(x, y) \in S$. For high-dimensional spaces, the distances $\delta(c, z)$ can be computed in constant time (i.e. in $O(1)$, instead of $O(d)$) purely from the distances $\delta(x, y)$, $\delta(x, z)$, and $\delta(y, z)$ observing that x, y, z span a planar subspace which includes c and in which the law of cosines holds and therefore: $4\delta(c, z)^2 = 2\delta(x, z)^2 + 2\delta(y, z)^2 - \delta(x, y)^2$ (cf. Figure 3(b)). Furthermore, splitting an edge only affects the join tree if the density at its mid-point is lower than the minimum density of its end-points. We only split an edge when needed, thus keeping the size of the graph small and allowing us to terminate the calculation of f for that mid-point once it exceeds the minimum.

The second problem can be solved by considering only a subset of the Delaunay triangulation. The Gabriel graph is considered a good approximation of the Delaunay graph with respect to neighboring vertices. However, this graph is not simplicial, therefore a precondition of the join tree algorithm is not met. Yet, this condition is only sufficient and not necessary, as it is trivial to construct simplicial complexes for which the deletion of some edges does not alter the output of the algorithm. In comparison to the Delaunay graph, the Gabriel graph omits the long edges of simplices. The density along these omitted edges is usually lower than the densities along the preserved edges, therefore the saddles of the join tree are unlikely to occur on removed edges. This is fortunate, as we are only interested in the join tree and therefore make only a small mistake by considering only the Gabriel edges. This graph can be computed using $O(n^3)$ distance computations. As this can be still a very long time, we present our improved Gabriel graph computation in Appendix A.

The mesh construction requires all points to be distinct, but in practice this requirement may not be met. We therefore construct the unique point set from the original data points and remember for each unique point the set of data points it represents.

4.2 Computing the Topology

Given the Gabriel graph of P, augmented by the additional m vertices on the edges and the density at all vertices, we can perform the join tree graph algorithm from [9]. Because minima and split saddles are not of interest for point clouds, we compute just the join tree. The join tree still contains all the information about maxima and the densities at which contours join. It thus preserves the number and nesting of dense regions. The algorithm itself is performed on a graph for which we trivially bound the number of edges by $n(n-1)$ which in turn bounds the run-time of the join tree computation by $O(n^2 \log n)$.

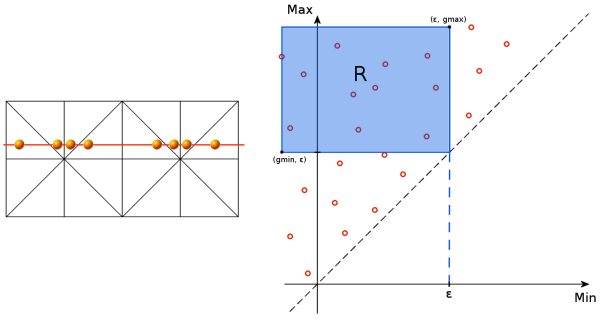
Using a trivial modification of the join tree algorithm, we can remember, for each input graph vertex that represents a data point, on which join tree edge it lies. Since join tree edges are reflected by regions in the topological landscape, this mapping allows for a proper placement of data points in the landscape.

4.3 Generating the Landscape

To visualize the join tree, we essentially make use of the landscape metaphor as introduced by Weber *et al.* [36]. To improve the resolution of the generated landscape, they present a method they term "rebalancing" to simplify the nesting of hills in the input contour tree which we apply to our trees as well. Before rebalancing, we remove topological noise by pruning branches from the branch decomposition which exhibit a persistence smaller than a given percentage of the main branch's persistence. These two simplifications are subject to one parameter each.

Weber *et al.* [36] also present a way to distort the landscape so that the size of features in the original space is visible. We apply this distortion too, but as we cannot approximate the volume of single elements in the high dimensional space we use the number of data points on each join tree edge as the desired area of the corresponding landscape region. [36] lacks details about the actual smoothing of the landscape. We could not find a suitable triangle-based interpolation that would preserve the topology of the landscape, therefore we join triangles back into quads and use Hermite interpolation on them. The resulting quads are split back again into triangles.

We augment the landscape by inserting the data points as small spheres in suitable locations. Unfortunately, the original distance information gets mostly lost by computing the topology and can not be used. However, because each data points lies on exactly one branch of the join tree and the topology of the landscape equals the join tree there must be exactly one contour on that branch's hill for that point's density. We place each data point at a random position along this contour. Since the contours are not present explicitly, but have to be computed from the triangle structure of the landscape, local accumulation of the data spheres can arise if not treated appropriately. Figure 4(a) shows some triangles which share a given height value, marked by the red height line. Assuming a sphere that should be placed at that height, each point on the red line would be topologically correct. Obviously, a sequential or random choice of a triangle (and then a position inside this triangle) would result in accumulations of data spheres at those positions where neighboring triangles intersect only a small portion of the contour. To avoid this, we select a random triangle using probabilities that are proportional to the amount of triangle/contour intersection. This results in



(a) (b)

Figure 4: (a) For a given height value, a sequential or random choice of a triangle using uniform probabilities results in accumulation artifacts. Therefore, we use probabilities proportional to the amount of intersection of each triangle with the contour of that height. (b) Points in the *span space* represent min-max height values of triangles. Only the points in $R = [gmin, \varepsilon] \times [\varepsilon, gmax]$ are related to triangles that share a given height value ε.

a much more equally distributed arrangement of the spheres.

For each data point, we need to compute the set of triangles that are intersected by that point's contour. Testing each triangle becomes expensive if the landscape contains many triangles, which occurs especially after smoothing. For example, by means of subdivision surfaces or parametric surfaces, each pyramid-like hill (and the place in between) can be smoothed to a realistic hill, resulting in a huge amount of triangles. To boost the search process, we make use of some ideas from the field of accelerated isosurface extraction [28]: Each triangle is mapped to a point (min, max) where *min* and *max* denote that triangles minimum and maximum height value, respectively. In this so called *span space*, all triangles that "span" over a given height value ε are contained in the rectangle $R = [gmin, \varepsilon] \times [\varepsilon, gmax]$ where *gmin* and *gmax* denote the minimum and maximum height of all triangles (see Figure 4(b)). To determine these points $k_i \in R$ efficiently, the span space is organized in a kd-tree, which allows a quick range query using R. The construction of a kd-tree is known to take $O(n \log n)$ and a range query takes $O(\sqrt{n} + |k|)$ [11]. We use such a kd-tree for each branch separately to quickly determine the set of triangles that intersect a data point's contour.

4.4 Reading the Landscape

One remaining question, when applying the method to a given data set, is the determination of the filter radius. A filter radius that is too large results in a rather blurry density distribution, thus combining actually separated dense regions. Likewise, a filter radius that is too small creates many peaks in the density distribution, thus identifying dense regions where there are none, e.g. at small noise accumulations. The analyzing process therefore consists of a sequential application of the method, using different filter radii until the best has been found, e.g. the number of maxima does not change for a significant range of the filter radius σ. For this task, the structure of the landscape and the arranged data points can be used to determine the next filter radius. Starting with a particular radius, e.g. the diameter of the data's hyper-volume or the longest distance between any two points, several pieces of information can be obtained by *reading* the landscape:

1. *The number of hills*
   Unless the data set actually contains only one dense region, the presence of one single hill (or only a few hills) is an indicator of the filter radius being too large. Many small hills with only a few data points on it indicate a filter radius too small.

2. *The data point distribution on a hill*
   If dense regions are indeed separated, the densities of data points belonging to the dense region are quite different from the density of the saddle. Therefore, these points are arranged somewhere on the upper half of the hill. If many points are arranged at the foot of the hill, they are either noise or the filter radius is too small, thus splitting a single dense region into several ones.

3. *The data point separation on a hill*
   If the data points on a hill constitute *rings* of different height, then the filter radius is too big and combines dense regions of different density.

Using these observations, the filter radius for the next step can be determined manually. An example for this process is given in Figure 7. Note that the algorithm does not have to recalculate everything. The pairwise distances and the Gabriel graph can be reused. The density function, the join tree and the topological landscape have to be generated again.

5 Experiments

We implemented our method in C++ and with the exception of the join tree and the topological landscape computation, all phases of our algorithm were parallelized. We tested our algorithm with several data sets on a computational platform with two 2.6GHz Quad-Cores and 8GB memory. Our biggest example required 2.9GB.

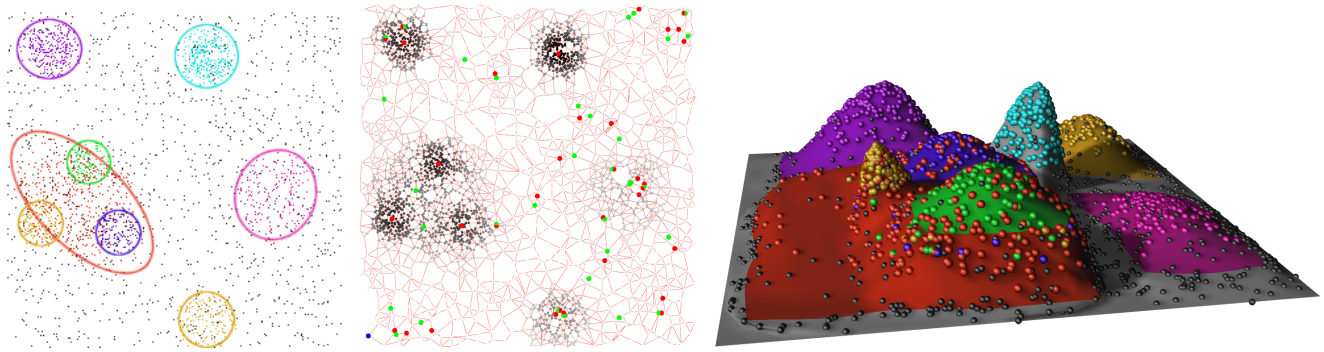
5.1 Two Dimensional

While the great benefit of the new approach is that we can visualize the approximated topology of point clouds in arbitrary dimensions, we first want to explain a simple two-dimensional example. This is just to get a feeling of how features in the data correspond to features in the landscape and how the Gabriel graph in general looks like.

As a toy example, we consider a simple artificial two-dimensional data set of 2,748 points, shown in Figure 5(a). The data consists of several dense regions (one of which is nested) and the color of each point represents its cluster relationship. Furthermore, the dense regions have different size, shape and density (i.e. points per area) and we heavily afflicted the data with noise (black points) to subvert the regular dense regions.

We start with the computation of the Gabriel graph which is given in Figure 5(b). It contains 6,222 edges which is 0.14% of the number of edges the complete graph would have. We then compute the density at each vertex. The result is also shown in Figure 5(b) by means of color coding at the graph vertices. Darker points correspond to higher densities. Subsequently, edges are split when the density at their midpoint is lower than at their vertices. In our example, this happens 203 times, resulting in a graph with 2,951 vertices and 6,425 edges. The join tree is calculated next. In Figure 5(b), we marked maxima red, join saddles green and the global minimum blue. As can be seen, each dense region hosts one density maximum and between them there are single saddles. There are several extremum-saddle pairs, which have a rather small persistence. They are considered to be topological noise and are pruned.

We compute the join tree's topological landscape and arrange the data points at their appropriate positions. The result can be seen in Figure 5(c). Since the cyan cluster is the one with the highest density, it hosts the global density maximum, thus it constitutes the extremum of the root branch and, being the highest hill, is centered on the landscape. The footprints of the hills reflect the number of vertices on their corresponding branches in the join tree, i.e. the number of data points in the dense regions. Consequently, the red



(a) (b) (c)

Figure 5: (a) The synthetic example data set (the colored borders are just for better visual identification) consisting of some colored clusters afflicted with noise. (b) The Gabriel graph with the color-coded density distribution (dark=dense) and the critical points of the *join-tree* (red=maximum, green=saddle, blue=minimum). (c) The resulting topological landscape, after pruning and rebalancing using a threshold of 10% of the maximum density.

hill and the gray part (belonging to the root branch) are the biggest ones. Furthermore, the nesting structure of the dense regions is reflected by the nesting structure of the hills and the noise points are spread over the whole landscape (as they are spread over the whole data set). Finally, except from the red points, which are mixed with the orange, green and blue points, the colored points are solely on their hills. The overall computation time for this example, i.e. from reading the data till the final landscape, was approximately three seconds.

5.2 Ten Dimensional

In Figure 7, we demonstrate the effect of different filter radii on the topological landscapes of a 10-dimensional data set[1]. The overall computation time for this example, containing 1,200 data points, was less than 2 seconds for each image. The Gabriel graph contained 18,260 edges which is 2.53% of the number of edges in the complete graph. For the four values of σ, the edge splitting operations performed were 2, 1,298, 3,205 and 4,722. Generally, lower values of σ result in more edges being split.

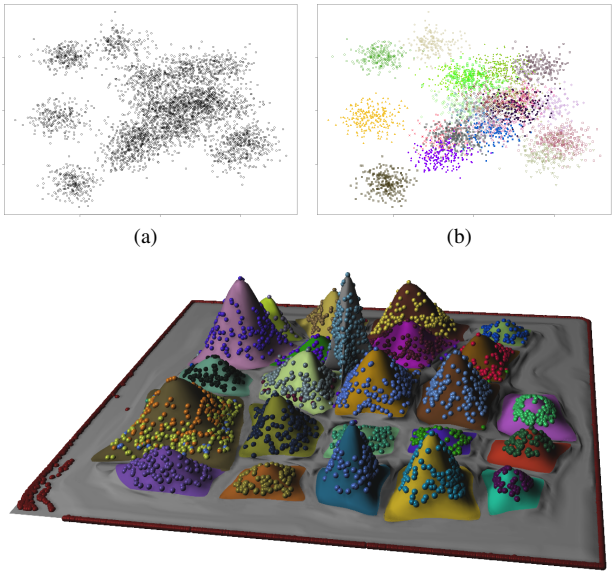
5.3 High Dimensional

As a high dimensional example, we decided to use a sufficiently complex example, produced by a generator described in [19]. It uniformly arranges normal distributions in a high dimensional space. We generated a 60-dimensional data set containing 25 clusters of random radius. In order to complicate the whole scenario, we added the same number of noise points as were produced by the clustering generator. This gave us a point cloud having a total of 8,452 points.

For comparison with other methods, we initially apply some PCA to get a clue about the data. Figure 6(a) shows the projection of the input data onto the first two principal components. Note that the additional noise points were ignored here to allow for a useful scatterplot. Including the noise, the actual clustering would be hidden by a big containing noise-cloud. Although the PCA roughly indicates a clustering, the projection discards too much information about the data's structure and neglecting the coloring (which is casually unlikely in real data sets) leads to at most around 10 clusters. Also, if actually available, the clustering-based coloring of the PCA in Figure 6(b) does not reveal the actual point cloud's structure.

However, it is good enough for a first glimpse and we therefore apply our new method to the data. To find an appropriate filter radius, we used the approach described in Section 4.4. After some

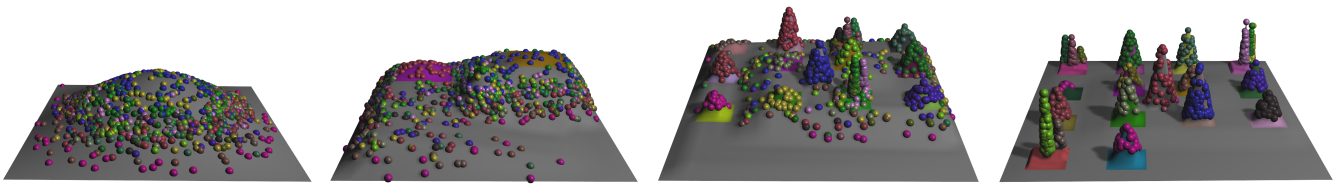
[1]In particular, it is the data set "10d-20c-no3.dat" from the dataset collection, accessible at *http://dbkgroup.org/handl/generators/*



(a) (b) (c)

Figure 6: (a)-(b) The projection of the data points (without noise) onto the first two principal components coarsely reveals the clustering structure of the data set. (c) The Topological Landscape clearly shows 24 separated hills which correspond to the dense regions. The extremely sparse noise comprises a ring at the height of low density, thus indicating that all noise points are each on their own.

passes with different radii, we end up with the landscape shown in Figure 6(c). First of all, the landscape clearly possesses a couple of separated hills, indicating the separated maxima in the density distribution. In particular, it has twenty-four of the possible twenty-five hills, meaning that two clusters are still combined by the currently used filter radius. Moreover, the data points on the hills are separated as well. We want to note here that a hill on a hill does not necessarily mean that the corresponding dense regions are completely enclosing each other. Structured hills indicate on the one hand that the saddle between the corresponding regions is higher than densities at the boundary, and on the other hand these regions are closer to each other than to other regions (i.e. other hills on the landscape). The noise in high dimensional spaces is



(a) (b) (c) (d)

Figure 7: The user *reads* the landscape in order to determine the correct filter radius: (a) Starting with an obviously too big filter radius, we see a single hill, harboring rings of different height and color; (b) Reducing the filter radius, the first hills start to occur, representing the local maxima in the density distribution; (c) A further reduction of the filter radius leads to more hills, while some points from different clusters are still mixed on some hills and other points are still considered as being noise; (d) Finally, all the data points are separated on their corresponding hills; a further reduction would result in new local maxima inside the clusters, i.e. additional, smaller hills harboring a few points which could be pruned from the branch decomposition due to low persistence.

extremely sparse, thus being most likely outside the clusters. This sparseness leads to very small (and most likely equal) densities. Therefore, the corresponding spheres, which of course belong to the root branch, constitute a ring at the height between the global minimum (i.e. the height of the landscape's boundary) and the saddle value of the root branch's last child branch. Of course, this depends on the data set. As shown in the first example, the noise could also be mixed with the dense regions, thus taking every possible density value. In this case, the corresponding spheres would be placed at their appropriate positions. Regarding the run-time, the overall computation time for this rather large and high dimensional example was about 377s. The amount of Gabriel graph edges compared to the complete graph was 28.1%. Note that even without the coloring, the same information could have been extracted from the topological landscape.

5.4 General Observations

We made some observations regarding the algorithm's general behavior in higher dimensions:

1. *The Gabriel graph approaches the complete graph*

   If the number of points is fixed, the number of relative neighbors increases with every additional dimension the points span and the Gabriel graph's vertices have an increasing number of neighbors. This has an effect on the topology which becomes increasingly trivial. In the end, the topology is not distinguishable from a single dense region. To counter this effect, the number of points should always be sufficiently larger than the number of dimensions they span.

2. *The distance computation takes more time*

   With each additional degree of freedom, the costs to determine the Euclidean distance (i.e. the scalar product) increases, as well. This affects on the one hand the need for additional memory to store the high dimensional vectors and on the other hand the duration of distance calculations.

3. *Noise points are on their own*

   With each additional degree of freedom, a fixed amount of data points becomes increasingly *sparse*. Moreover, when dimensionality increases, data points are likely located in different dimensional subspaces, which in turn affects the meaning of distance, because these points can all be considered as being equally distanced. These phenomena are known as *curse of dimensionality* and are widely discussed since they affect almost every algorithm in higher dimensions.

Regarding these observations, some things are important for higher dimensions. First of all, with each additional dimension the runtime and memory complexity increases. Because we deal to some extend with $O(n^2)$ data structures and algorithms, this might constitute a bottle neck for higher dimensions, but also depends on the data itself. Secondly, matters like the *curse of dimensionality* and the ratio between data points and their dimensionality, force the Gabriel graph to become more full.

6 Conclusion and Future Work

We presented a multiple stage process to achieve an approximation of the topology of high dimensional point clouds. This approximation aims on the study of the data's structure in terms of dense regions, thus considering the number, size and nesting structure of point accumulations. We therefore describe the data indirectly by its density distribution and visualize a 3-D topological landscape, having the same (*joining*-)topology as this distribution.

To automatically present the best possible landscape, our future work focuses on the automatic choice of the filter radius. Furthermore, we analyze methods to apply our approach only to a limited number of dimensions, i.e. by analyzing only subspaces. Although, we actually want to describe the whole situation, i.e. by considering all dimensions, this might speed up the overall process by neglecting unimportant information in some dimensions.

Acknowledgements

We thank Hamish Carr for valuable discussions regarding our method and Gunther H. Weber for assisting the realization of our topological landscape implementation. We would also like to thank the anonymous reviewers for their useful comments. This work was supported by a grant from the German Science Foundation (DFG) under number SCHE663/4-1 within the strategic research initiative on Scalable Visual Analysis (SPP 1335).

A Improved Gabriel Graph Computation

For large number of points the Gabriel graph computation can be very slow. There are optimizations for the two and three dimensional case, but they do not extend to higher dimensions. Instead, we present the following improvements: in Euclidean spaces each triplet of (non-collinear) points u, v, w uniquely defines a plane, and the intersection of u, v's Gabriel lune with the plane is a circle with center c. Because of Thales' theorem which holds in this plane, the test whether w is outside the Gabriel lune can be written as $\delta(u,v)^2 \leq \delta(u,w)^2 + \delta(v,w)^2$. Using this method we no longer need to compute the circle centers and can determine the Gabriel graph using just the pairwise distances between the original point set. Of course, these distances can be computed beforehand reducing the run-time from $O(dn^3)$ to $O(dn^2 + n^3)$. The inner loop of the naive Gabriel graph algorithm aborts earlier if the points of P are tested in increasing distance from u or v. Furthermore, points w cannot lie within the Gabriel lune if $\delta(u,w)^2 + \Delta(v)^2 \geq \delta(u,v)^2$ and

$\Delta(v)$ denotes v's distance to its nearest neighbor. Using these simple ideas we can greatly reduce the number of iterations in the inner loop if we sort the neighbors of each point for increasing distance. The final algorithm is presented in the following:

Algorithm 1 Pseudo-code of improved Gabriel graph computation

Require: a point set $P = (p_1, p_2, \dots p_n)$
Ensure: the Gabriel graph G

```
compute all pairwise distances δ(p_i, p_j)
determine each point's nearest neighbors distance Δ(p_i)
sort P for decreasing Δ(p_i)
G ← (P, ∅)
for i = 1 to n do
    sort P for increasing distance from p_i into P'
    for j = 1 to i − 1 do
        skipedge ← false
        for x ∈ P' with δ(p_i,x)^2 < δ(p_i,p_j)^2 − Δ(p_j)^2 do
            if δ(p_i,x)^2 + δ(p_j,x)^2 < δ(p_i,p_j)^2 then
                skipedge ← true and abort loop
            end if
        end for
        if skipedge = false then
            G ← G + (p_i, p_j)
        end if
    end for
end for
```

We observed that the innermost loop was finished on average after 5–10 iterations for our data sets. For lower dimensions, because a point inside the lune was found quickly and in higher dimensions the length of the loop decreased dramatically due to the "curse of dimensionality". The run-time of the algorithm becomes dominated by distance calculations and sorting. Note that each point can still be treated independently. It is therefore possible to compute using a maximum of n processors with varying workloads in parallel.

REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, 28(2), 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data. *Data Min. Knowl. Discov.*, 11(1):5–33, 2005.

[3] B. Alpern and L. Carter. The hyperbox. In *IEEE VIS '91*, pages 133–139, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[4] K.-H. Anders. *Parameterfreies hierarchisches Graph-Clustering Verfahren zur Interpretation raumbezogener Daten (Dissertation)*. PhD thesis, Universitaet Stuttgart, 2003.

[5] M. Ankerst, D. Keim, and H.-P. Kriegel. Circle segments: A technique for visually exploring large multidimensional datasets. In *Proceedings Visualization '96*, 1996.

[6] R. A. Becker and W. S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[7] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[8] R. L. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *AFIPS '63 (Fall): Proceedings of the November 12-14, 1963, fall joint computer conference*, pages 445–458. ACM, 1963.

[9] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 2003.

[10] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey, editors. *Graphical methods for data analysis*. The Wadsworth Statistics/Probability Series, 1983.

[11] M. De Berg, O. Cheong, and M. van Kreveld. *Computational geometry: algorithms and applications*. Springer, 2008.

[12] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[13] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[14] N. Elmqvist, P. Dragicevic, and J.-D. Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1539–1148, 2008.

[15] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[16] S. Fortune. *Voronoi Diagrams and Delaunay triangulations*, pages 193–233. World Scientific Press, 1992.

[17] G. W. Furnas and A. Buja. Prosection views: Dimensional inference through sections and projections. *JCGS*, 3:323–385, 1994.

[18] R. K. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, 1969.

[19] J. Handl and J. Knowles. Cluster generators for large high-dimensional data sets with large numbers of clusters, 2005, http://dbkgroup.org/handl/generators.

[20] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[21] A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *VIS '90: Proceedings of the 1st conference on Visualization '90*, pages 361–378, 1990.

[22] G. Jaromczyk, J.W.; Toussaint. Relative neighborhood graphs and their relatives. In *Proc. of the IEEE*, volume 80 of *Issue 9*, 1992.

[23] I. T. Jolliffe. *Principal component analysis*. Springer, Berlin, 1986.

[24] D. A. Keim, M. Ankerst, and H.-P. Kriegel. Recursive pattern: A technique for visualizing very large amounts of data. In *VIS '95: Proc. of the 6th conference on Vis. '95*. IEEE Computer Society, 1995.

[25] T. Kohonen. *Self-Organizing Maps*, volume 30. Springer Series, 2001.

[26] J. B. Kruskal and M. Wish. *Multidimensional Scaling (Quantitative Applications in the Social Sciences)*. SAGE Publications, 1978.

[27] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[28] Y. Livnat, H.-W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[29] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[30] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. Multi-resolution computation and presentation of contour trees. In *Lawrence Livermore National Laboratory, Tech. Rep. UCRL-PROC-208680*, 2005.

[31] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, pages 428–439, 1998.

[32] S. Takahashi, I. Fujishiro, and M. Okada. Applying manifold learning to plotting approximate contour trees. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1185–1192, 2009.

[33] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[34] E. Tufte. *The Visual Display of Quantitative Information, 2nd. Ed.* Graphics Press, 2001.

[35] W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB*, pages 186–195, 1997.

[36] G. Weber, P.-T. Bremer, and V. Pascucci. Topological landscapes: A terrain metaphor for scientific data. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1416–1423, 2007.

[37] G. H. Weber, G. Scheuermann, and B. Hamann. Detecting critical regions in scalar fields. In *IEEE TCVG Symposium on Visualization, pp. 1-11*, 2003.