Online Data Visualization of Multidimensional Databases Using the Hilbert Space–Filling Curve

Jose Castro and Steven Burns

Costa Rica Institute of Technology
jose.r.castro@gmail.com, royalstream@gmail.com

Abstract. We propose in this paper a visualization approach for large online databases using the Hilbert space–filling curve to map N–dimensional data points to 2D or 3D points. Dimensionality reduction methods like principal component analysis (PCA), multi dimensional scaling (MDS) or self organizing maps (SOMS) can map N–dimensional data points with N>>3 into 3 dimensional or 2 dimensional values that allow us to visualize the data. These methods although popular, require either the calculation of a scatter matrix, eigenvalues and eigenvectors, or the iteration of learning algorithms. Therefore these methods cannot perform online, can be slow with large databases and always produce information loss when the data is mapped from the multidimensional space to the 2D or 3D image. Space–filling curves like the Peano, Z, and Hilbert curve, on the contrary, produce a 1–to–1 mapping between points in a line segment and an arbitrary N–Dimensional hypercube. This 1–to–1 mapping guarantees that there is no information loss on the transformation. Specifically the Hilbert space–filling curve is known to preserve the Lebesgue measure and has been proven to produce an optimal mapping in the sense that an arbitrary contiguous block of information will receive the minimum number of splits in the mapped space. The Hilbert space–filling curve has been extensively used for indexing and clustering by mapping N–dimensional data points to 1–dimensional values. We propose here to use the curve to map to 2 or 3 dimensions for purposes of visualization: By taking advantage of its 1–to–1 nature, a new and generic method to map N–dimensional data points to 2D or 3D points using the Hilbert space–filling curve is developed. We prove theoretically that the calculation of the mapping can be done in constant time if we fix the order of approximation, thereby giving linear O(n) performance on the number of data points to map. We create a Hilbert space–filling curve visualization tool that is much faster than the other methods mentioned and allows us to generate quickly for very large datasets various different visualizations of the data, thereby compensating the lack of use of statistical information in the calculation of the mapped points. We compare our approach to MDS and PCA with a benchmark data set and three real datasets using the distance preserving and topology preserving measure as benchmarks. Our experiments indicate that the Hilbert space–filling curve produces acceptable quality of mapping while achieving much faster visualization and is therefore especially useful for online visualization of very large data sets.

P.P. Lévy et al. (Eds.): VIEW 2006, LNCS 4370, pp. 92–109, 2007.

1 Introduction

Dimensionality reduction methods like principal component analysis (PCA) (Duda et al 2000), multi dimensional scaling (MDS) or self organizing maps (SOMS) (Estévez et al 2000) have been extensively used to map N–dimensional data points with N>>3 into 3 dimensional or 2 dimensional values. These techniques have many uses, one of them being the visualization of complex multidimensional data.

Although quite popular, PCA and related approaches require either the calculation of a scatter matrix, eigenvalues and eigenvectors, or the iteration of time consuming learning algorithms, and as a consequence degrade considerably when the number of dimensions and/or data points grows. Another consequence of applying these techniques is that the learning algorithm or PCA matrix compresses the data in a loss-full transformation when the data is mapped from the multidimensional space to the 2D or 3D image (this loss is intentional, since PCA and learning algorithms are meant to extract the meaningful information from the data and eliminate the superfluous features). Also the use of population information for the mapping like the scatter matrix makes it difficult, if not impossible, to create an online visualization of the data points.

In this paper we propose the use of space–filling curves like the Peano, Z, and Hilbert curve for the purpose of data visualization. This approach, like any other, has advantages and disadvantages, and these algorithm trade offs should be known by the visualizer to be able to take good advantage of the technique. We particularly test the Hilbert space–filling curve using the topology preserving measure and the distance preserving measure.

This paper is organized as follows: In the second section we review quickly the principles behind the statistically based techniques and the patter recognition techniques that we will be comparing to (PCA and Sammon Mapping). In the third section we introduce the space filling curves and their different variants focusing specifically on the Hilbert space–filling curve (HSFC). This section ends with a detailed description for the HSFC algorithm that we developed that is a simplified version of Butz original algorithm (Butz 1968). The computation complexity of this method is linear (optimal) with respect to the number of patterns to draw in the data set. Section four explains the distance preserving and topology preserving comparison methods we used and then proceeds in explaining the test sets. Section five presents and discusses the results of our experimentation. We end with conclusions and indications of further research directions that we intend to pursue.

2 Principal Component Analysis

The Karhunen-Loeve transform or Principal Components Analysis (PCA) is a well known linear orthogonal transform widely used in data projection and pattern recognition. In PCA we create a transform that is optimal in a sum squared error sense. PCA first calculates the mean $\bar{\mathbf{x}}$ and scatter matrix S for the data set. Once this

is done we calculate the eigenvalues and eigenvectors $\lambda_k, \mathbf{e}_k$ of *S*. Here, we make a selection of which eigenvectors to use for mapping. Usually the values of the eigenvalues decay rapidly, indicating that some of the dimensions in the multidimensional space comprise noise in the data. For visualization, we select the 2 or 3 largest eigenvalues and their corresponding eigenvectors and create a $d \times k$ matrix **A**, where d is the number of dimensions and k is the number of chosen eigenvectors (2 or 3). The transformation will be:

$$\mathbf{x}' = \mathbf{A}^t(\mathbf{x} - \bar{\mathbf{x}}) \tag{1}$$

2.1 Sammon Mapping and the SAMANN Network

Sammon Mapping (Sammon, 1969) is a useful procedure to reduce the dimensionality of a data set, preserving as well as possible the inter-pattern distances from the original input points.

The distance measure (D) more commonly used is the Euclidean distance and the error function to be minimized is the following:

$$E = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} D(\mathbf{x}_i, \mathbf{x}_j)} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{\left(D(\mathbf{x}_i, \mathbf{x}_j) - D(\mathbf{y}_i, \mathbf{y}_j)\right)^2}{D(\mathbf{x}_i, \mathbf{x}_j)} \tag{2}$$

That error function is known as the *Sammon Stress* function and Sammon Mapping minimizes it using standard gradient descent or second order methods.

Instead of the standard Sammon Mapping, we will use a Neural Network implementation: SAMANN (Mao, 1995.)

The SAMANN is usually a two layer network with sigmoid activation functions whose input layer has one neuron for each dimension in the input set and the output layer has one neuron for each dimension in the output set. The network is fed two patterns at a time and its trained based on the distance between them with a special learning rule. Please refer to the original paper for more details (Mao, 1995.).

3 Space-Filling Curves and the Hilbert Curve

The discovery of space-filling curves is credited to Peano (1890), when he found a continuous curve that visited every point of a closed square exactly once. A space-filling curve S^n can be considered as a mapping from the unit hypercube $[0,1]^n$ into the unit interval $[0,1]$.

Space–filling curves in general produce a 1–to–1 mapping between points in a line segment and an arbitrary N–Dimensional hypercube (Mokbel et al 2002). This 1–to–1 mapping guarantees that there is no information loss on the transformation.

Specifically the Hilbert space–filling curve is known to preserve the Lebesgue measure and has been proven to produce an optimal mapping in the sense that an

arbitrary contiguous block of information will receive the minimum number of splits in the mapped space (Moon et al 2000). The Hilbert space–filling curve has been extensively used for indexing and clustering by mapping N–dimensional data points to a 1–dimensional values (Lawder 2000, Mokbel 2004). We propose here to use the curve to map to 2 or 3 dimensions for purposes of visualization:

By taking advantage of it's 1–to–1 nature, we can map data points from an arbitrary *d*–dimensional space to a 3 dimensional or 2 dimensional space in two steps: first, for every point **x** in the data set, we map it to its one dimensional Hilbert index **x'**. Once this is done we use this **x'** and map it to a value $\mathbf{y} \in [0,1]^3$ using a Hilbert inverse mapping (if we are visualizing in 3 dimensions). This mapping does not lose information because it's 1–to–1 and therefore has an inverse mapping.

In real life, the space-filling curves used are in fact approximations that only visit a finite subset of points by limiting the order of approximation of the curve, but our experience is that with as little as ten bits per dimension we already have a fine enough mapping to be able to give unique image values to every point even for large databases (+2 Million points).

Formally, The m^{th} order approximation curve, denoted by S_m^n, has a grid size of 2^m, and maps a total of 2^{mn} points from an n-dimensional space into a scalar value. The grid size is the number of divisions into which each dimension is split. The actual space filling curve is the limit of this sequence of curves.

$$S^n = \lim_{m \to \infty} S_m^n \tag{3}$$

Hilbert generalized the definition to an arbitrary number of dimensions and provided a general geometric procedure to construct them. There are many other space-filling curves like the z-curve, the gray curve, etc. However, as shown by (Mokbel et al 2004), the Hilbert space-filling curve produces the least number of splits in an index, as a product of being continuous and devoid of jumps or biased towards any dimension.

The first 3 approximations of the Hilbert curve for a 2-dimensional space can be seen in Figure 1.

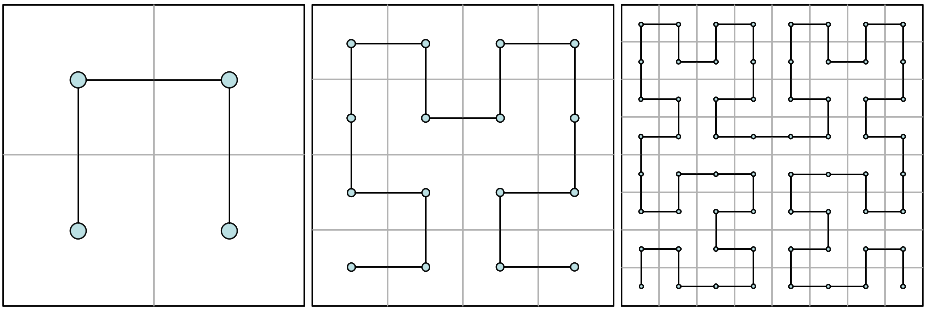

Fig. 1. First 3 approximations for the Hilbert curve in 2 dimensions

3.1 The Hilbert Mapping Algorithm

There are different algorithms for calculating the Hilbert mapping. The one presented is a slight simplification of (Lawder 2000), which in turn is a modification of an iterative algorithm originally presented by (Butz 1968).

The algorithm maps numbers in binary representation and the precision achieved is determined by the order of approximation employed. The output scalar and coordinates of the input vector are real values in [0,1[but the algorithm uses integer variables only with the first bit representing the first bit after the decimal point, so the number 0.11010001 for example would get represented as the integer 11010001.

Please note that all the sub-indexes used in the algorithm are zero-based to ease the programming in any language with zero-based arrays such as C or C++[1].

$$
\begin{aligned}
&Hilbert(\langle a_0, a_1, ..., a_{n-1} \rangle, m)\\
&\quad \varpi, \tau, J_{tot} \leftarrow 0\\
&\quad for\ i \in [0, m-1]\\
&\qquad \varpi \leftarrow \varpi \otimes \tau\\
&\qquad \sigma \leftarrow \alpha^i \otimes \varpi\\
&\qquad \sigma \leftarrow \sigma \lll_{\circ} J_{tot}\\
&\qquad r^i \leftarrow \sigma \otimes (\sigma \gg 1) \otimes (\sigma \gg 2) \otimes ... \otimes (\sigma \gg (n-1))\\
&\qquad J \leftarrow \text{principal position of } r^i\\
&\qquad \tau \leftarrow \sigma \text{ complemented at position } (n-1)\\
&\qquad if\ \tau \text{ has odd parity}\\
&\qquad\quad \tau \leftarrow \tau \text{ complemented at position } J\\
&\qquad endif\\
&\qquad \tau \leftarrow \tau \ggg_{\circ} J_{tot}\\
&\qquad J_{tot} \leftarrow J_{tot} + J\\
&\quad endfor\\
&\quad r \leftarrow 0.r^0 r^1 ... r^{m-1}\\
&return\ r
\end{aligned}
$$

Where:

n is the number of dimensions.

m is the order of approximation.

[1] Please refer to the Appendix for sample code in C++ for the mapping algorithms.

r is the scalar output of the algorithm (the mapped value) in the range [0,1] represented in $n \cdot m$ bits.

r^i is the i^{th} word of *n* bits from r with $i \in [0, m-1]$ such that $r = r^0 r^1 ... r^{m-1}$.

a_j is the j_{th} coordinate of the input vector $\langle a_0, a_1, ..., a_{n-1} \rangle$ being mapped represented in *m* bits, with $j \in [0, n-1]$.

α^i is a word of *n* bits whose j_{th} bit has the same value as the i^{th} bit of a_j.

$\omega, \sigma, \tau, J, J_{tot}$ are auxiliary variables of the algorithm with no special meaning.

Parity: number of bits of a word whose value is non-zero.

Principal Position: If all the *n* bits of the word have the same value, the principal position is by definition $n-1$. Otherwise, it's the zero-based index of the rightmost bit whose value differs from the bit in the last position (*n*-1).

Finally, $\ll$ and $\gg$ are the standard bit-shift operators, $\ll\circ$ and $\circ\gg$ represent circular bit-shift operations and $\otimes$ stands for the bitwise XOR.

The reverse mapping is the exact same algorithm executed in reverse and it's omitted for the sake of brevity. Nevertheless, the source code for the mapping in both directions is included in the Appendix.

The algorithm is clearly $O(nm)$ which is a clear advantage over the other techniques presented.

4 Comparison Measures

In order to compare numerically the different mapping procedures we use two different measures. The former quantifies the topology preservation of the data whereas the latter measures the conservation of the distances. Both measures provide numerical values for comparison and analysis.

4.1 Topology Preservation

The Topology Preservation Measure (Andreas 2000) provides a way to quantify the conservation of the local neighborhood for each element in the data set.

The index of the i^{th} nearest neighbor for a given pattern $\mathbf{x}_j$ in the original high-dimensional space gets denoted by $NNX(j,i)$. Therefore $\mathbf{x}_{NNX(j,1)}$ would be the nearest neighbor for $\mathbf{x}_j$.

Following the same notation, the i^{th} nearest neighbor for the corresponding pattern $\mathbf{y}_j$ in the low-dimensional space is $NNY(j,i)$. The following credit scheme is then applied:

$$qm_{ji} = \begin{cases} 3 & \text{if } NNX(j,i) = NNY(j,i) \\ 2 & \text{if } NNX(j,i) = NNY(j,t) \quad t \in [1,n],\ t \neq i \\ 1 & \text{if } NNX(j,i) = NNY(j,t) \quad t \in]n,k],\ n < k \\ 0 & \text{else} \end{cases} \tag{4}$$

Basically, each one of the *n* nearest neighbors of $\mathbf{x}_j$ gets a score between 0 and 3. The highest score means their relative position was preserved exactly by the mapping. The following score applies if their position changed but stayed within the neighborhood of the nearest *n* elements. Finally, the lowest non-zero score applies if the element is found in a broader neighborhood of *k* elements. Usual values for *n* and *k* are 4 and 10 respectively (Andreas 2000, Estévez et al 2005).

Summing the scores through the whole dataset and dividing by a normalizing factor we obtain the Topology Preservation Measure:

$$qm = \frac{1}{3nN} \sum_{j=1}^{N} \sum_{i=1}^{n} qm_{ji} \tag{5}$$

As it can be seen from the equation above, the measure is a real number between 0 and 1, where $qm = 1$ would indicate a perfect preservation of the topology for the given parameters.

The topology preservation measure ignores the explicit values of the Euclidean distances and only cares about their relative ordering. Therefore, it's invariant to translations, rotations and uniform rescaling of all coordinates of the data set in one or both dimensional spaces.

4.2 Distance Preservation: Sammon Stress

This measure can be interpreted as an error or penalty assigned to the differences between the distances in the original space and the mapped space, see (2). Note that the first part of the equation is a constant normalizing factor that can be calculated beforehand.

Since this measure is based solely on distances between the points, it's insensitive to translations of the data set and it's said to be invariant to uniform rescaling (D. Ridder 1997).

However, from (2) it can be seen that the measure remains invariant only if both sets of points in both dimensional spaces get rescaled equally and simultaneously. Therefore for a constant set of points in the original high-dimensional space, multiple rescaled versions of given mapping would yield different Sammon Stress measures.

Some dimensionality reduction techniques such as the Hilbert curve or the SAMANN (with sigmoid outputs) produce mapped patterns whose values are restricted to the unitary interval. Rescaling the input data set is not an option because it's not known beforehand how large could get the mapped coordinate values (D. Ridder 1999) and this limits the minimum Sammon Stress they can obtain.

After the mapping process is finished, we propose scaling the output data set by a factor β that minimizes the stress function:

$$\beta = \arg\min_{\beta} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{\left(D(\mathbf{x}_i, \mathbf{x}_j) - \beta D(\mathbf{y}_i, \mathbf{y}_j)\right)^2}{D(\mathbf{x}_i, \mathbf{x}_j)} \tag{6}$$

Solving that equation we find the appropriate scaling factor to be applied to avoid unnecessary penalties caused by scale differences:

$$\beta = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} D(\mathbf{y}_i, \mathbf{y}_j)}{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{D(\mathbf{y}_i, \mathbf{y}_j)^2}{D(\mathbf{x}_i, \mathbf{x}_j)}} \tag{7}$$

The calculation of this factor we propose is $O(n^2)$ which is no different than the calculation of the Sammon Stress itself.

5 Experiments

Three real-world databases of three different sizes were used. For the bigger databases random subsets were taken to run the experiments instead of using the database as a whole because the comparative measures employed, specially the Sammon Stress, are quadratic with respect to the number of cases.

1. *Iris Database:* The Iris database from the UCI Machine Learning Repository is used extensively in pattern recognition and it's also referenced by most of the referenced works about dimensional mapping. It contains information about 150 iris plants of three different classes: Setosa, Versicolor and Virginica. Each element is represented by four different attributes: petal length, petal width, sepal length and sepal width. This database is rather small and we include it solely for comparative purposes.

2. *Letter Image Recognition Data:* Also from the UCI MLR, contains data about the 26 capital letters in the English alphabet. Each letter was rendered in one out of 20 possible fonts and distorted randomly afterwards. The data set contains 20000 different patterns with 16 attributes relative to their position, pixels, dimensions and their means, variances and correlations. Three different data sets of 1000, 2000 and 5000 records were randomly selected to perform the experiments.

3. *Forest Covertype:* From the Department of Forest Sciences of the Colorado State University, contains information about 581012 trees with seven different classes of cover type. The original database presented 54 attributes but number 11 to 14 were mutually exclusive booleans that could get represented as a single integer between

1 and 4. Attributes 15 to 54 presented the same behavior and were also reduced to a value between 1 and 40 leaving us with a total of 12 attributes for this database. Just like before, the experiments were performed over three random data sets of 1000, 2000 and 5000 records.

The high-dimensional data sets were mapped onto a low-dimensional space using the Hilbert Space-filling curve. First each pattern vector is reduced to a single scalar value using the described algorithm. Arbitrary-precision integers were employed in the code preventing any loss of information and allowing this operation to be completely reversible. The scalar value is then expanded using the reverse algorithm to a vector in the 2 or 3-dimensional space for visualization.

This methodology is not employed by any of the reviewed references. It differs from the traditional approach (Keim 1995, Wettenberg 2005) where the Hilbert curve is used to arrange the points of one single dimension into a rectangular area or sub-window, later each dimension gets its own sub-window on the screen and there is no dimensionality reduction taking place whatsoever.

For each one of the data sets, 50 independent runs were performed with a different random order of the dimensions. The Hilbert curve is not biased towards any dimension (Moon et al 2001) therefore similar results are expected from all of them.

For each run, both 2-dimensional and 3-dimensional mappings were created and the Topology Preservation Measure was calculated along with the Sammon Stress after scaling the mapped data according to (7).

Principal Components Analysis was run on the same data sets and the same measures were also calculated for its 2-dimensional and 3-dimensional mappings. PCA is our main point of comparison through the experiments.

Finally, we created a SAMANN network and initialized its first layer with the eigenvector matrix as shown by (Lerner et al 2000). It was run for all the data sets with 1000 or less points and only for a two-dimensional projection. The training was stopped when the change in the error was less than 0.00001 or when a time limit had elapsed.

5.1 Experimental Results

The summarized results for the mappings using the Hilbert curve can be seen in the Table 1. For each of the 50 runs of each data set the best value, the mean and the standard deviation of both quality measures are presented. As expected from the properties of the curve, the standard deviations are low.

The results for the PCA are listed in the Table 2. It gives better topology preservation than the Hilbert curve for smaller databases like the Iris, but underperforms for the Letter-recognition database and the Cover-type database in the two dimensional cases.

Also, for those two bigger databases the quality of the topology preservation measures for PCA seems to decrease as the sample size gets larger. All the statistics of the PCA are based on the $d \times d$ covariance matrix which seems to become less representative of the nature of the data as the number of instances grows.

For the Hilbert mapping the topology preservation doesn't decrease as the size of the data sets increases. Actually for the Letter Recognition database the best topology preservation was obtained when using the biggest sample size of 5000 rows. The obtained values are also surprisingly similar for the 2D and 3D cases.

Table 1. Experimental results using the Hilbert Mapping

Dataset	Sample size	Dims	Topology P.M.			Sammon Stress		
			Best	Mean	StDev	Best	Mean	StDev
Iris	150(all)	2D	0.451	0.430	0.012	0.218	0.338	0.066
Letter-rec	1000	2D	0.222	0.200	0.009	0.233	0.307	0.043
Letter-rec	2000	2D	0.223	0.208	0.006	0.227	0.302	0.042
Letter-rec	5000	2D	0.243	0.234	0.004	0.234	0.305	0.040
Cover-type	1000	2D	0.282	0.271	0.006	0.168	0.392	0.088
Cover-type	2000	2D	0.276	0.266	0.005	0.245	0.386	0.093
Cover-type	5000	2D	0.269	0.262	0.004	0.239	0.382	0.104
Iris	150(all)	3D	0.446	0.416	0.017	0.201	0.281	0.042
Letter-rec	1000	3D	0.224	0.200	0.009	0.185	0.228	0.025
Letter-rec	2000	3D	0.225	0.208	0.007	0.182	0.227	0.030
Letter-rec	5000	3D	0.243	0.234	0.004	0.179	0.223	0.025
Cover-type	1000	3D	0.289	0.272	0.008	0.191	0.311	0.069
Cover-type	2000	3D	0.279	0.267	0.004	0.156	0.300	0.061
Cover-type	5000	3D	0.271	0.264	0.004	0.152	0.293	0.065

Table 2. Experimental results using PCA

Dataset	Sample	Dims	TPM	SS
Iris	150(all)	2D	0.558	0.009
Letter-rec	1000	2D	0.118	0.202
Letter-rec	2000	2D	0.102	0.198
Letter-rec	5000	2D	0.080	0.198
Cover-type	1000	2D	0.267	0.075
Cover-type	2000	2D	0.211	0.074
Cover-type	5000	2D	0.165	0.072
Iris	150(all)	3D	0.782	0.001
Letter-rec	1000	3D	0.262	0.108
Letter-rec	2000	3D	0.243	0.110
Letter-rec	5000	3D	0.205	0.109
Cover-type	1000	3D	0.342	0.045
Cover-type	2000	3D	0.295	0.044
Cover-type	5000	3D	0.262	0.043

PCA shows better performance for the 3D case which was expected because as the number of dimensions in the mapped space grows, the output data set approaches the original data (but on different coordinate axes).

As far as the Sammon Stress, both mapping techniques perform similarly for the Letter recognition database in 2 dimensions, but PCA seems to preserve the distances better for the rest of the cases.

The Hilbert mapping seems to create clusters of points that are very close together and usually share the same class. This could favor class separability but hurts the distance preservation measure.

Figures 2 to 7 show different visualizations obtained by both mapping methods.

As far as the SAMANN network, we found that it gets stuck very easily in local minima and as (D.Ridder 1997) mentions, they are slower and harder to train than ordinary ANNs and there are full papers devoted to its initialization. Nevertheless, for the Iris database it obtained a topology preservation of 0.34 and a Sammon Stress of 0.043, clearly a local-minimum as it can be appreciated in the Figure 8.

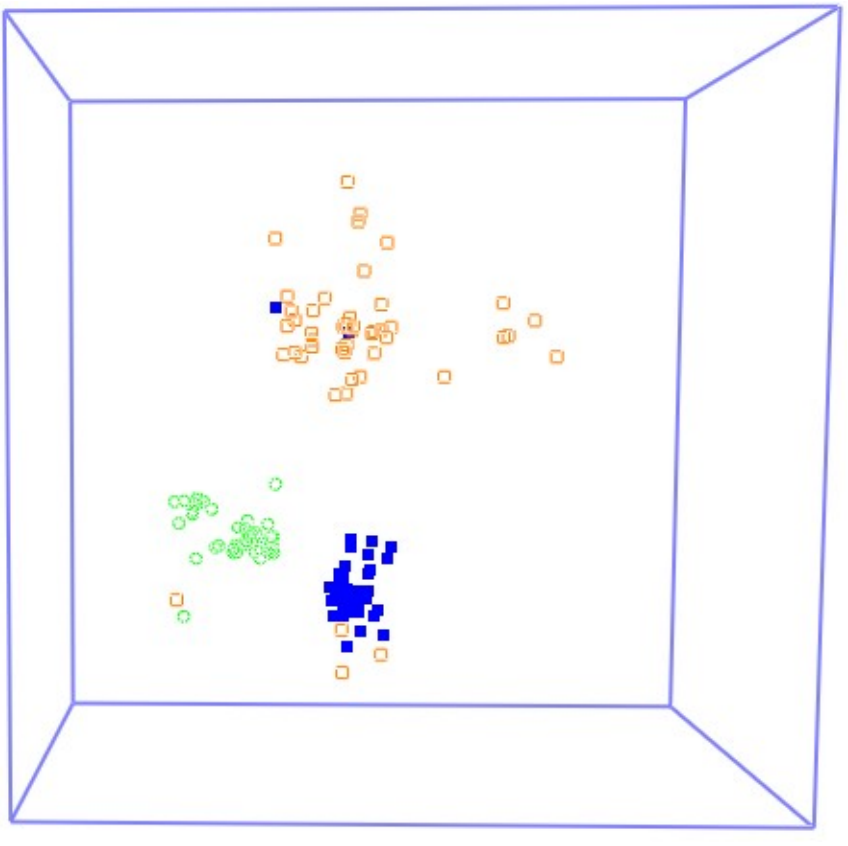

Fig. 2. Hilbert 3D visualization of the Iris database

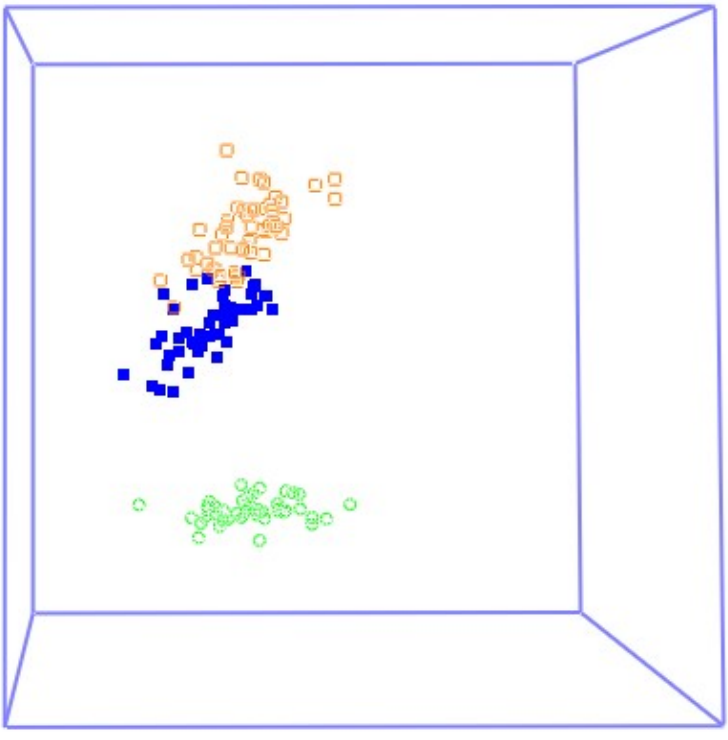

Fig. 3. PCA 3D visualization of the Iris database

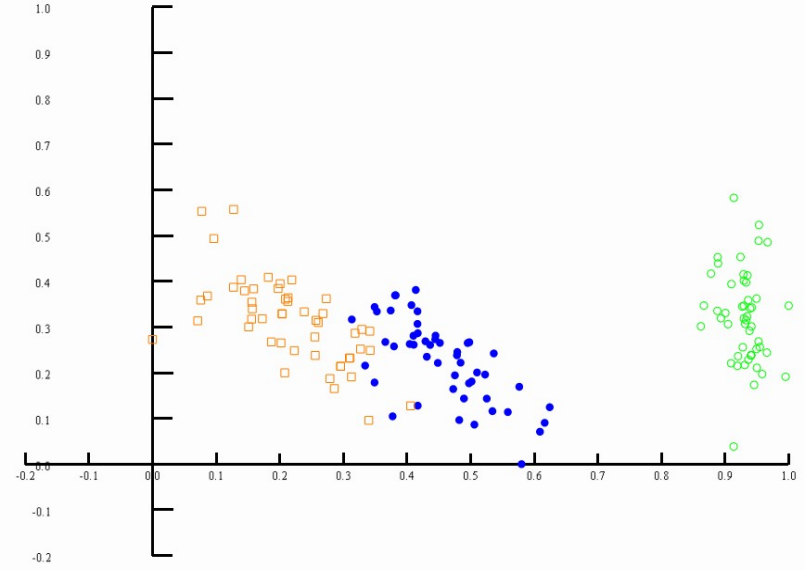

Fig. 4. PCA 2D visualization of the Iris database

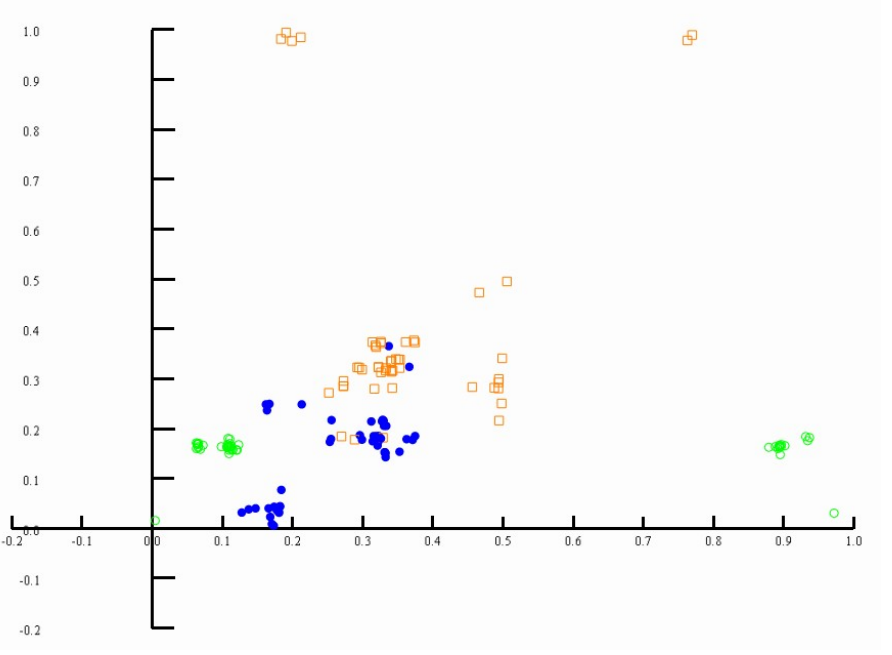

Fig. 5. Hilbert 2D visualization of the Iris database

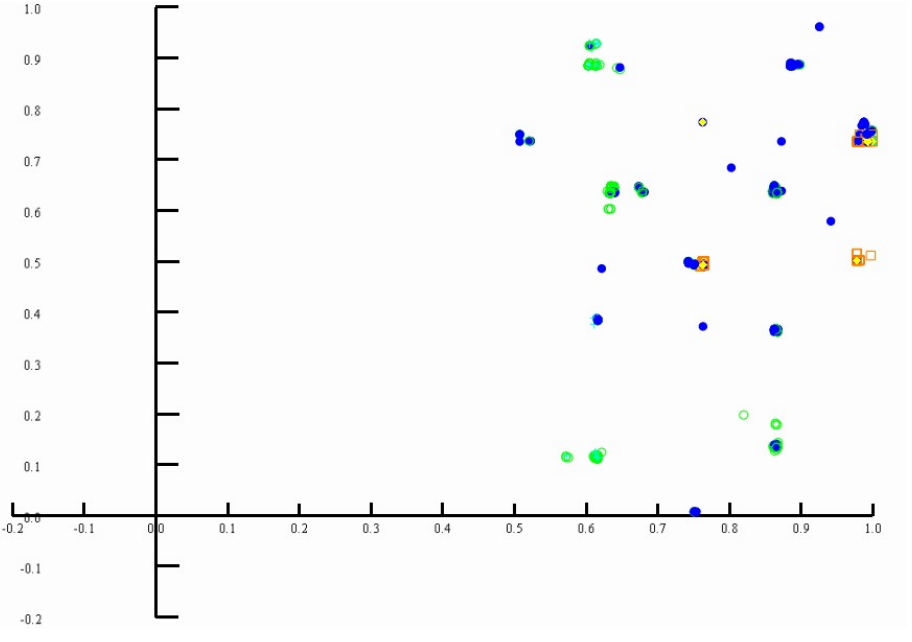

Fig. 6. Hilbert 2D visualization of 1000 points from the Cover-type database

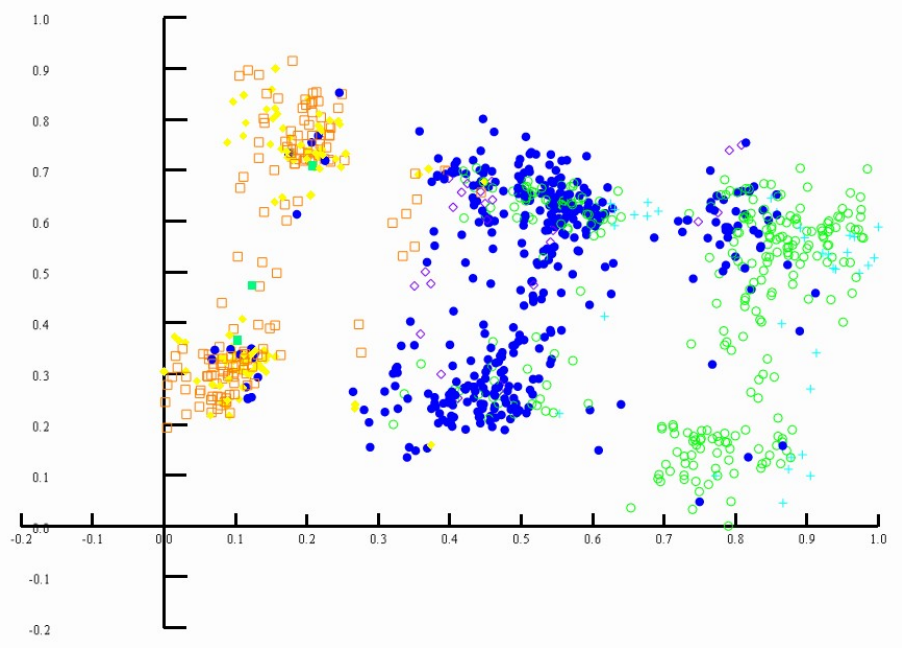

Fig. 7. PCA 2D visualization of 1000 rows from the Cover-type database

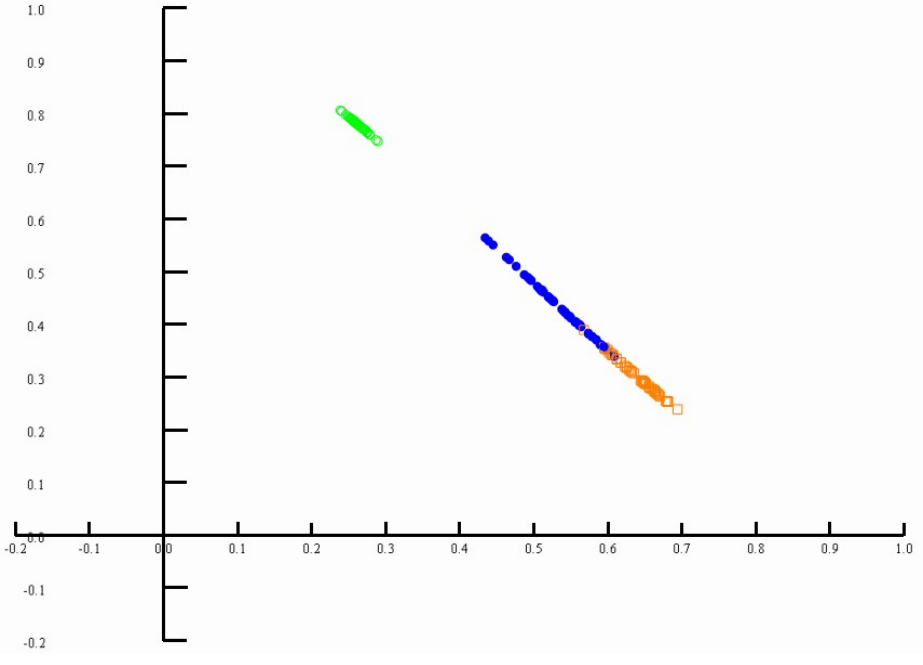

Fig. 8. SAMANN local-minimum visualization for the Iris database

6 Conclusions and Future Work

The Hilbert Curve provides acceptable visualizations at a very small computational cost. The algorithm is linear with respect to both, the number of elements in the data set and the number of dimensions.

Its quantitative performance seems to favor the topology preservation and not so much the distance preservation measure (Sammon Stress). Also, the first measure presented almost no variance for the different runs with different random order of dimensions whereas the Sammon stress presented higher differences.

As an additional contribution, we propose a modification to the distance preservation measure in which the data is rescaled uniformly by a factor that minimizes the error function, allowing us to compare different mapping methods regardless of the output scaling.

As the number of records used in the input data set grows, the Hilbert mapping seems to deal better with the bigger data sets than PCA, this seems to indicate it's better suited to handle large databases than the traditional methods. Large databases also benefit in terms of speed since the mapping is done with integer shift and bitwise operations and the algorithm is linear to both the number of dimensions and the size of the data set.

It's also suited for online databases in which new cases or patterns appear and need to be visualized quickly because the method doesn't involve any recalculation of any statistics and the visualization of each point is independent of the other points in the set.

As a weakness, the mapping using the Hilbert curve treats all the dimensions in the input data set equally which could degrade the quality of the mapping and the visualization if the data set contains noisy or irrelevant dimensions. However, given the simplicity and speed of the algorithm, users could experiment including or excluding certain dimensions as the visualization process takes place.

Visually, most of the projections made with the Hilbert curve show clusters of points of the same class grouped closely. PCA, on the other hand, attempts to preserve the variance of the projected data and this explains the better scores on the Sammon Stress.

Future work might include the use of additional mapping quality measures which take into consideration the separation of the different classes in the projected mapping. Furthermore, this paper is part of a wider ongoing research regarding the applications of the space-filling curves not only to visualization but to classification problems and clustering.

References

(Butz 1968) A. R. Butz. *Space filling curves and mathematical programming*. Information and Control, 12:314-330, 1968.

(Duda et al 2000) Richard O. Duda, P. E. Hart, David G. Stork. *Pattern Classification*. Wiley & Sons. 2nd Edition. 2000.

(Estévez et al 2005) Pablo A. Estévez,; Cristián J Figueroa; Kazimo Saito. *Cross-Entropy Approach to Data Visualization Based on the Neural Gas Network*. IJCNN 2005, Montreal Canada.

(Keim 1995) Keim, D. *Enhancing the visual clustering of Query-Dependent Database Visualization Techniques Using Screen-Filling Curves,* Lecture Notes In Computer Science, Vol. 1183, Springer-Verlag, London.

(König 1998) Andreas König. *A Survey of Methods for Multivariate Data Projection, Visualisation and Interactive Analysis*. Dresden University of Technology, Germany, 1998.

(König 2000) Andreas König. *Interactive visualization and Analysis of Hierarchical Neural Projections for Data Mining*. IEEE Transactions on Neural Networks, Vol. 11, No. 3, May 2000.

(Lawder 2000) J. K. Lawder. *Calculations of Mappings Between One and n–dimensional Values Using the Hilbert Space–filling Curve*. Technical Report no. JL1/00, August 15, 2000.

(Lerner et al (A) 1999) B. Lerner, H. Guterman, M. Aladjem, and I. Dinstein. *A comparative study of neural network based feature extraction paradigms*. Pattern Recognition Letters, vol. 20, no. 1, pp. 7-14, 1999.

(Lerner et al 1996) B. Lerner and H. Guterman and M. Aladjem and I. Dinstein. *Feature Extraction by Neural Network Nonlinear Mapping for Pattern Classification*. ICPR13, Vienna, vol. 4, 320-324, 1996.

(Lerner et al (B) 1999) B. Lerner, H. Guterman, M. Aladjem, I. Dinstein, and Y. Romem. *On pattern classification with Sammon's nonlinear mapping - an experimental study*. Pattern Recognition, vol 31, pp. 371-381, 1998.

(Lerner et al 2000) B. Lerner and Hugo Guterman and Mayer Aladjem and Its'hak Dinstein. *On the Initialisation of Sammon's Nonlinear Mapping*. Pattern Analysis and Applications, Springer-Verlag, London, 2000.

(Mao 1995). Mao, A.K. Jain, *Artificial neural networks for feature extraction and multivariate data projection.* IEEE Trans. Neural Networks 6, 1995.

(Mokbel et al 2002) Mohamed F. Mokbel, Walid G. Aref, Ibrahim Kamel. *Performance of Multi Dimensional Space–Filling Curves*. Proceedings of the 10th ACM symposium on Advances in geographic information systems. ACM Press, New York, USA. 2002.

(Mokbel et al 2004) Mohamed F. Mokbel, Walid G. Aref. Ibrahim Kamel. *Fast and effective characterization of 3D Regions of Interest in medical image data*. Medical Imaging 2004: Image Processing, Proceedings of SPIE Vol 5370. 2004.

(Moon et al 2001) Bongki Moon, H. V. Jagadish, Christos Faloustos, Joel J. Saltz. *Analysis of the Clustering Properties of the Hilbert space–filling Curve*. IEEE Transactions on Knowledge and Data Engineering, Vol 13. No 1. January/February 2001.

(Pekalska et al 1999) Elzbieta Pekalska, Dick de Ridder, Robert P.W. Duin, Martin A. Kraaijveld. *A new method of generalizing Sammon mapping with application to algorithm speed-up*. Delft University of Technology, The Netherlands, 1999.

(de Ridder et al 1997) Dick de Ridder and Robert P. W. Duin. *Sammon's mapping using neural networks: A comparison*. Pattern Recognition Letters, 18:1307-1316, 1997.

(Wattenberg 2005) Martin Wattenberg. *A Note on Space-Filling Visualizations and Space-Filling Curves*. INFOVIS.

Appendix: Sample Source Code

The following C++ source code fragment includes the functions required to perform the dimensional mappings from N-dimensional to 1-dimensional values and back. Please read the remarks section at the end of the source code for additional information.

```
// N-dimensional value to 1-dimensional value:
UINT n_to_one(const UINT a[], int dims, int order)
{
    UINT Wi = 0;
    UINT Ti = 0;
    UINT r = 0;
    int Jtot = 0;
    for(int i = 0; i < order; i++)
    {
        UINT Ai = 0;
        for(int j = 0; j < dims; j++)
            if(is_bit_on(a[j],i))
                Ai = set_bit_on(Ai, j);
        Wi = Wi ^ Ti;
        UINT Oi = Ai ^ Wi;
        Oi = left_shift_circular(Oi, Jtot, dims);
        UINT Yi = Oi;
        for(int j = 0; j < (dims-1); j++)Yi ^= (Oi >> (j+1));
        r |= clean_up(Yi,dims) >> (i * dims);
        int J = get_principal(Yi,dims);
        Ti = swap_bit(Oi,dims-1);
        if(has_odd_parity(Ti,dims))Ti = swap_bit(Ti,J);
        Ti = right_shift_circular(Ti, Jtot, dims);
        Jtot += J;
    }
    return r;
}
  // 1-dimensional value back to N-dimensional space:
void one_to_n(UINT r, int dims, int order, UINT a[])
{
    for(int j = 0; j < dims; j++)a[j] = 0;
    UINT Ti = 0;
    UINT Wi = 0;
    int Jtot = 0;
```

```
    for(int i = 0; i < order; i++)
    {
        Wi = Wi ^ Ti;
        UINT Yi = r << (i * dims);
        UINT Oi = Yi ^ (Yi >> 1);
        int J = get_principal(Yi,dims);
        Ti = swap_bit(Oi,dims-1);
        if(has_odd_parity(Ti,dims))Ti = swap_bit(Ti,J);
        Oi = right_shift_circular(Oi, Jtot, dims);
        Ti = right_shift_circular(Ti, Jtot, dims);
        UINT Ai = Wi ^ Oi;
        for(int j = 0; j < dims; j++)
            if(is_bit_on(Ai,j))
                a[j] = set_bit_on(a[j], i);
        Jtot += J;
    }
}
```

Remarks

- The `UINT` type found in the code could be a `typedef` to the platform's 64-bit unsigned integer or simply to `unsigned int`, but in any event the size of the type imposes a limitation to the number of dimensions (and bits per dimension). For the general case an arbitrary-precision C++ class is required, overloading the shift/bitwise operators and functions accordingly. All our experiments were executed this way.
- This sample code requires the implementation of several bitwise functions such as: `is_bit_on`, `set_bit_on`, `swap_bit`, `has_odd_parity`, `get_principal`, `clean_up`, `left_shift_circular` and `right_shift_circular`.
- The last five functions listed require the number of dimensions as a parameter because they need to know how many bits are being used within each `UINT` value.
- The `clean_up` function is required to set the remaining (unused) bits to zero when they could interfere with the calculations.