RIVA: Indexing and Visualization of High-Dimensional Data via Dimension Reorderings

Michail Vlachos[1], Spiros Papadimitriou[1],
Zografoula Vagena[2], and Philip S. Yu[1]

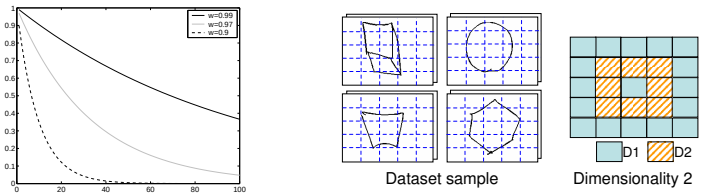
[1] IBM T.J. Watson Research Center, Hawthorne, NY, USA
[2] IBM Almaden Research Center, San Jose, CA, USA

Abstract. We propose a new representation for high-dimensional data that can prove very effective for visualization, nearest neighbor (NN) and range searches. It has been unequivocally demonstrated that existing index structures cannot facilitate efficient search in high-dimensional spaces. We show that a transformation from points to sequences can potentially diminish the negative effects of the dimensionality curse, permitting an efficient NN-search. The transformed sequences are optimally reordered, segmented and stored in a low-dimensional index. The experimental results validate that the proposed representation can be a useful tool for the fast analysis and visualization of high-dimensional databases.

1 Introduction

Suppose that we are interested in performing search operations on a set of high-dimensional data. For simplicity let us assume that the data lie in a unit hypercube $C = [0, 1]^d$, where d is the data dimensionality. Given a query point, the probability P_w that a match (neighbor) exists within radius w in the data space of dimensionality d is given by $P_w(d) = w^d$. Figure 1(a) illustrates this probability for various values of w. Evidently, at higher dimensionalities the data becomes very sparse and even at large radii, only a small portion of the entire space is covered. This is also known as dimensionality curse, which in simple terms translates into the following fact: for large dimensionalities existing indexing structures outperform sequential scan only when the dataset size (number of objects) grows exponentially with respect to dimensionality.

In this work we propose a mapping from high-dimensional to low-dimensional spaces that will boost the performance of traditional indexing structures—such as R-trees—without changing their inner-workings, structure or search strategy. This mapping will essentially condense the sparse/unused data space by grouping and indexing together dimensions that share similar characteristics. We will accomplish this by applying the following transformations: i) Conceptually, we will treat high-dimensional data as ordered sequences. ii) The original D dimensions will be reordered to obtain a *globally* smooth sequence representation. This will lead to placement of dimensions with similar behavior at adjacent positions in the ordered representation as sequence. iii) The resulting sequences will



(a) (b)

Fig. 1. (a) Probability $P_w(d)$. (b) Mapping of 25-D image features onto 2 dimensions and correspondence of projected against original dimensions.

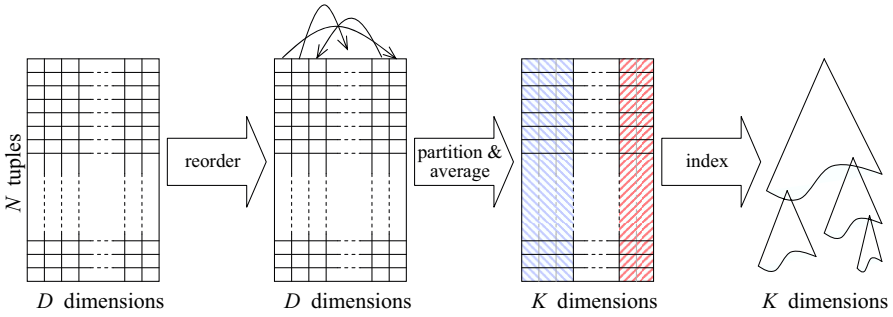


Fig. 2. Summarization of steps.

be segmented into groups of $K < D$ dimensions which can be then stored in a K-dimensional indexing structure. The previous steps are illustrated in Figure 2.

Elements of our techniques are partially inspired by or adapted from concepts in parallel coordinates visualization [4], time-series representation [9], co-clustering and bi-clustering [8] methodologies. However, the final goal is distinct from the previous techniques, since the focus of this work is primarily on the indexing of high-dimensional data. We note, however, that since our approach also relies on the efficient grouping of correlated/co-regulated attributes, the proposed algorithms can also be utilized for the identification of the principal data axes for high-dimensional datasets.

In Figure 1(b) we demonstrate an example of the dimension grouping and dimensionality reduction achieved by our techniques. The dataset consists of 25-dimensional features extracted from multiple images using a 5×5 grid (more details are provided in the experimental section). Each image belongs to one of the following four shape classes: *cube, ellipse, hexagon and trapezoid*. The shapes are drawn by humans, so they exhibit dislocations or distortions and no two images are identical. Using the proposed low dimensional projection/grouping, we map each 25-dimensional point onto 2 dimensions. We depict the correspondence between sets of original dimensions and each of the projected dimensions. Observe that peripheral and center parts of the image (which correspond to almost empty pixel values) are collapsed together into one projected dimension. Similarly centrally located portions of the image are also grouped together to form the second dimension. While this example illustrates the usefulness of our dimension grouping techniques for image/multimedia data, we should emphasize the utility of our methods in a number of other domains:

Table 1. Description of main notation.

Symb.	Description	Symb.	Description
N	Database size (number of points).	$\mathcal{D}$	An ordering of all D dimensions.
D	Database dimensionality.	K	Number of dimension partitions.
$\mathbf{t}_i$	Tuples (row vectors), $\mathbf{t}_i \in \mathbb{R}^D$.	$\mathcal{B}$	Set of partition breakpoints.
$t_i(d)$	The d-th coordinate of $\mathbf{t}_i$.	$\mathcal{D}_k$	The k-th ordered partition.
$\mathbf{T}$	Database, as a $N \times D$ matrix.	D_k	Size of $\mathcal{D}_k$.

1. High-dimensional data visualization: Our technique intelligently groups of related dimensions, leading to an efficient low-dimensional interpretation and visualization of the original data. Our methods provide a direct mapping from the low-dimensional space to the original dimensions, permitting more coherent interpretation and decision making based on the low-dimensional mapping (contrast this with PCA, where the projected dimensions are not readily interpretable, since they involve translation and rotation on the original attributes).
2. Gene expression data analysis: Microarray analysis provides an expedient way of measuring the expression levels for a set of genes under different regulatory conditions. They are therefore very important for identifying interesting connections between genes or attributes for a given experiment. Gene expression data are typically organized as matrices, where the rows correspond to genes and columns to attributes/conditions. Our techniques could be used to mine either conditions that collectively affect the state of a gene or, conversely, sets of genes that are expressed in a similar way (and therefore may be jointly affecting certain variables of the examined disease or condition).
3. Recommendation systems: An increasing number of companies or online stores use collaborative filtering to provide refined recommendations, based on the historical user preferences. Utilizing common/similar choices between groups of users, companies like Amazon or Netflix can provide suggestions on products (or movies, respectively) that are tailored to the interests of each individual customer. For example, Netflix serves approximately 3 millions subscribers providing online rentals for 60,000 movies. By expressing rental patterns of customers as an array of customers versus movie rentals, our technique could be then used for identifying groups of related movies based on the historical feedback.

Summarizing the main contributions of this work: (i) We provide an efficient abstraction that can map high dimensional datasets into a low-dimensional space. (ii) The new space can be used to visualize the data on two (or three) dimensions. (iii) We demonstrate how the low dimensional space can be used in conjunction with existing indexing structures (such as R-trees) for mitigating the adverse effect of high-dimensionality on the index search performance. (iv) Finally, the proposed data mapping effectively organizes the data features into logical subsets. This readily allows for efficient determination of correlated or co-regulated data features.

2 Preliminaries

In the following sections we will describe our methodology for data reorganization which is called 'RIVA' (**R**eordering for **I**ndexing and **V**isu**A**lization). Assuming

a database $\mathbf{T}$ that consists of N points (rows) in D dimensions (columns), the goal is to reorder and partition the dimensions into K segments, $K < D$. We denote the database tuples as row vectors $\mathbf{t}_i \in \mathbb{R}^D$, for $1 \leq i \leq N$. The d-th value of the i-th tuple is $t_i(d)$, for $1 \leq d \leq D$. We begin by first defining an ordered partitioning of the dimensions. Then we introduce the necessary measures that characterize the quality of a partitioning, irrespective of order. Next, in Section 3 we will show how we can exploit reordering to find the partitions efficiently, with a single pass over the database.

Definition 1 (Ordered partitioning $(\mathcal{D}, \mathcal{B})$). *Let $\mathcal{D} \equiv (d_1, \ldots, d_D)$ be a* total ordering *of all D dimensions. The order along with a set of breakpoints $\mathcal{B} = (b_0, b_1, \ldots, b_{K-1}, b_K)$ defines an* ordered partitioning*, which divides the dimensions into K segments (by definition , $b_0 = 1$ and $b_K = D + 1$ always). The size of each segment is $D_k = b_k - b_{k-1}$. We denote by $\mathcal{D}_k \equiv (d_{k,1}, \ldots, d_{k,D_k})$ the portion of $\mathcal{D}$ from positions b_{k-1} up to b_k, i.e., $d_{k,j} \equiv d_{j-1+b_{k-1}}$, for $1 \leq j \leq D_k$.*

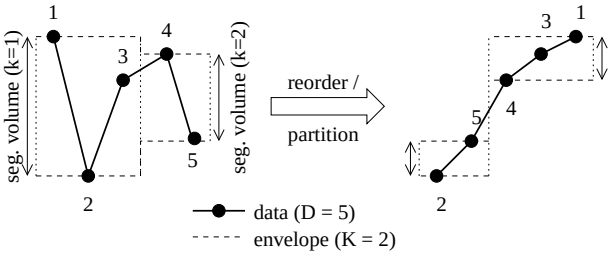


Fig. 3. K-dimensional envelopes of D-dimensional points. On the right, order is $\mathcal{D} = (2, 5, 4, 3, 1)$ with breakpoints $\mathcal{B} = (1, 3, 6)$ and partition sizes $D_1 = 2$ and $D_2 = 3$.

Next, we need a measure of quality. Given a partitioning, consider a single point $\mathbf{t}_i$. Ideally, we want the smallest possible variation among values of $\mathbf{t}_i$ within each partition $\mathcal{D}_k$. Figure 3 illustrates two different partitionings and their corresponding *envelopes* (dashed lines), which are simply the minimum and maximum values of $\mathbf{t}_i$ within each set of dimensions $\mathcal{D}_k$. The partition on the right has smaller *volume*.

Definition 2 (Envelope volume $v_i(\mathcal{D}, \mathcal{B})$). *The* envelope volume *of a point $\mathbf{t}_i$, $1 \leq i \leq N$ is defined by*

$$v_i(\mathcal{D}, \mathcal{B}) := \sum_{k=1}^{K} \big(\max_{d \in \mathcal{D}_k} t_i(d) - \min_{d \in \mathcal{D}_k} t_i(d)\big).$$

This is proportional to the average (over partitions) envelope width.

Definition 3 (Total volume $V(\mathcal{D}, \mathcal{B})$). *The* total volume *achieved by a partitioning is $V(\mathcal{D}, \mathcal{B}) := \sum_{i=1}^{N} v_i(\mathcal{D}, \mathcal{B})$*

We should point out that, although the width of an envelope segment $\mathcal{D}_k$ is related to the variance *within* that partition, the envelope volume v_i is different from the variance (over dimensions) of $\mathbf{t}_i$. Furthermore, the total volume V is not related to the vector-valued variance of all points, and hence is also not related to the per-column variance of $\mathbf{T}$, which is used in [2].

Summarizing, we shall seek a single partitioning of the dimensions for the entire database. To that end, we would like to minimize the total volume V.

3 Reordering

In the previous section we defined the notions of an ordered partitioning and of volume. Unfortunately, summation over all database points in V is the outermost operation. Hence, computing or updating the value of V would require buffer space kN for the minimum values and another kN for the maximum values, as well as $O(N)$ time. Since N is very large, direct use of V to find the partitioning is infeasible. Surprisingly, we can intelligently use the dimension ordering and recast the problem in a way that allows us to perform the search after a *single pass* over the database. The reordering of dimensions is chosen to maximize some notion of "aggregate smoothness" and serves two purposes: (i) provide an accurate estimate of the volume V without requiring $O(N)$ space and time, and (ii) locate the partition breakpoints. The next sections make these ideas precise.

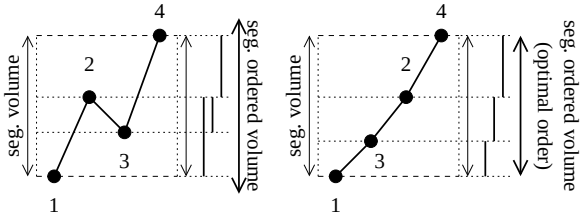


Fig. 4. Ordered volume for one data point, within a segment (see first segment in Figure 5), showing exactly the two volumes. Points on the right are in optimal order (see Lemma 1) and the ordered volume equals the "true" volume.

3.1 Volume through ordering

Consider a point $\mathbf{t}_i$ and a partition $\mathcal{D}_k$. Instead of the difference between the minimum and maximum over *all* values $t_i(d)$ for $d \in \mathcal{D}_k$, we will consider the sum of differences between *consecutive* values in $\mathcal{D}_k$.

Definition 4 (Ordered envelope volume $\bar{v}_i(\mathcal{D}, \mathcal{B})$). *The* ordered envelope volume *of a point* $\mathbf{t}_i$, $1 \leq i \leq N$ *is defined by*

$$\bar{v}_i(\mathcal{D}, \mathcal{B}) := \sum_{k=1}^{K} \sum_{j=2}^{D_k} |t_i(d_{k,j}) - t_i(d_{k,j-1})| = \sum_{\substack{j=1 \\ j \notin \mathcal{B}}}^{D} |t_i(d_j) - t_i(d_{j-1})|.$$

Figure 4 shows the ordered volumes of two different dimension orderings in one segment. The thin double arrows show the segment's volume. The thick lines on the right margin show the consecutive value differences. Their sum is the segment's ordered volume (thick double arrow).

Lemma 1 (Ordered volume). *For any ordering $\mathcal{D}$, we have $v_i(\mathcal{D}, \mathcal{B}) \leq \bar{v}_i(\mathcal{D}, \mathcal{B})$. Furthermore, holding $\mathcal{B}$ fixed, there exists an ordering $\mathcal{D}^*$ for which the above holds as an equality, $\bar{v}_i(\mathcal{D}^*, \mathcal{B}) = v_i(\mathcal{D}, \mathcal{B})$.*

The order $\mathcal{D}^*$ for which the ordered volume matches the original envelope volume of any point $\mathbf{t}_i$ is obtained by sorting the values of $\mathbf{t}_i$ in ascending (or descending) order—the full proof is ommited for space.

Definition 5 (Total ordered volume). *The* total ordered volume *achieved by a partitioning is* $\bar{V}(\mathcal{D}, \mathcal{B}) := \sum_{i=1}^{N} v_i(\mathcal{D}, \mathcal{B})$.

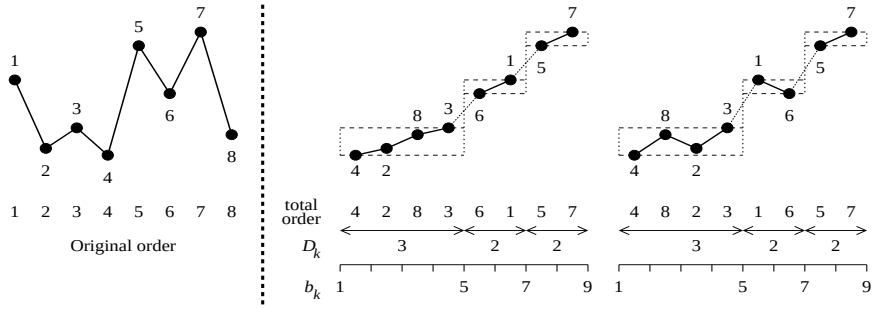


Fig. 5. One point and two total orders that correspond to the same partitioning ($D = 7$ and $K = 3$). The breakpoints b_k, $0 \leq k \leq K$ are also shown, with the induced partition sizes D_k, $1 \leq k \leq K$. The total ordering serves two purposes. First, to make the ordered volume within individual partitions close to the "true" volume. Second, to help us find the best breakpoints, which minimize the envelope and total volumes. The first reordering minimizes the sum of consecutive value differences, and achieves both goals.

Lemma 1 says that, for a given point $\mathbf{t}_i$, the ordering $\mathcal{D}$ allows estimation of the envelope volume using the sum of consecutive value differences. Furthermore, using a similar argument, we can show that a reordering $\mathcal{D}$ also helps us find the best breakpoints for a single point, i.e., the ones that minimize its envelope volume (see Figure 5—proof is straightforward but long and ommited for space).

Lemma 2 (Envelope breakpoints). *Let $\mathcal{D}^* \equiv (d_1, \ldots, d_D)$ be the ordering of the values of $\mathbf{t}_i$ in ascending (or descending) order. Given $\mathcal{D}^*$, let the breakpoints $b_1, \ldots, b_{K-1}$ be the set of indices j of the top-$(K-1)$ consecutive value differences $|t_i(d_j) - t_i(d_{j-1})|$ for $2 \leq j \leq D$. Then, $v_i(\mathcal{D}^*, \mathcal{B}^*) = \bar{v}_i(\mathcal{D}^*, \mathcal{B}^*)$ and this is the minimum possible envelope volume over all partitionings $(\mathcal{D}, \mathcal{B})$.*

3.2 Rewriting the volume

Next, we show that optimizing for $\bar{V}$, instead of V, can be performed with only a single pass over the database. By substituting the minimum and maximum operations (in v_i) with a summation (in $\bar{v}_i$), it is possible to exchange the summation order and make the summation over all points the innermost one. This allows us to compute this quantity once, hence requiring only a single scan of the database. First, we give a name to this sum.

Definition 6 (Dimension distance). *For any pair of dimensions, $1 \leq d, d' \leq D$, their* dimension distance *is the L^1-distance between the d-th and d'-th columns of the database $\mathbf{T}$, i.e.,$\Delta(d, d') := \sum_{i=1}^{N} |t_i(d) - t_i(d')|$.*

The dimension distance is similar to the consecutive value difference for a single point, except that it is aggregated over all points in the database. If some of the dimensions have similar values and are correlated, then we expect their dimension distance to behave similarly to the differences of individual points and have a small value. If, however, they are uncorrelated, we expect their dimension distance to be much larger. Now we can rewrite the expression for $\bar{V}(\mathcal{D}, \mathcal{B})$,

$$\bar{V}(\mathcal{D}, \mathcal{B}) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \notin \mathcal{B}}}^{D} |t_i(d_j) - t_i(d_{j-1})| = \sum_{\substack{j=2 \\ j \notin \mathcal{B}}}^{D} \Delta(d_j, d_{j-1}). \qquad (1)$$

3.3 Partitioning with TSP

With multiple points, we can no longer find the optimal ordering and breakpoints via a simple sorting. However, as observed before, sorting the values in ascending (or descending) order is equivalent to finding the order that minimizes the envelope volume and we can still seek an optimum of $\bar{V}$. As explained in Definition 6, we expect the dimension distance to behave similarly to the individual differences; it should be small for dimensions with related values and large for uncorrelated dimensions.

Instead of optimizing simultaneously for $\mathcal{D}$ and $\mathcal{B}$, we will first optimize for $\mathcal{D}$ and subsequently choose the breakpoints in a fashion similar to Lemma 2. Therefore, our objective function $C(\mathcal{D})$ is similar to Equation (1), except that it also includes dimension distances across potential breakpoints,

Definition 7 (TSP objective). *We will optimize for the* cost objective

$$C(\mathcal{D}) := \sum_{j=2}^{D} \Delta(d_j, d_{j-1}), \tag{2}$$

This formulation implies that $\Delta(d_1, d_D) \geq \Delta(d_j, d_{j-1})$, *for* $2 \leq j \leq D$.

If the last condition were not true, a simple cyclical permutation of $\mathcal{D}$ would achieve a lower cost. After we have found $\mathcal{D}^* = \arg\max_{\mathcal{D}} C(\mathcal{D})$, we will select the breakpoints in a fashion similar to Lemma 2, by taking the indices of the top-(K–1) dimension distances $\Delta(d_j, d_{j-1})$, $2 \leq j \leq D$. This simplification of optimizing first for $\mathcal{D}$ has the added benefit that we can very quickly try different values of K. But the objective of Equation (2) is that of the *traveling salesman problem (TSP)*, where nodes correspond to dimensions and edge lengths correspond to dimension distances. In Figure 6(a) the TSP tour is shown with thick lines. The breakpoints (for $K = 2$) are its two longest edges (dashed thick lines). The sketch of the RIVA algorithm is given below:

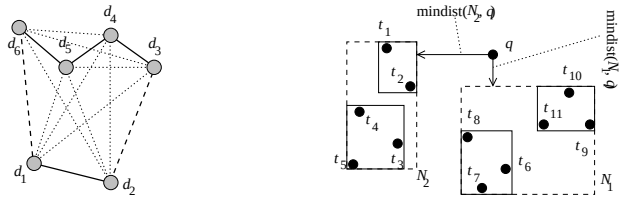
1. Scan the database once to compute the $D \times D$ matrix of dimension distances.
2. Find a TSP tour $\mathcal{D}$ of the D dimensions, using the above distances.
3. If necessary, rotate it to satisfy the condition in Definition 7.
4. Choose the remaining $K - 1$ breakpoints in $\mathcal{B}$ as described above.

The column reordering problem for binary matrices, which is a special case of the desired reordering for our problem is already shown to be NP-hard [5]. Finally, the dimension distance Δ satisfies the triangle inequality, in which case a factor-2 optimal of $C(\mathcal{D})$ can be found in polynomial time. In practice, even better solutions can be found quite efficiently (e.g., for $D = 100$, typical running time for TSP using Concorde[3] is about 3 seconds).

4 Indexing

In the previous section we outlined how to find an ordered partitioning that makes the points as smooth as possible, with a single pass over the database.

[3] `http://www.tsp.gatech.edu/concorde/`



(a) (b)

Fig. 6. (a) Illustration of TSP problem, (b) R-tree structure.

A natural low-dimensional representation of the points $\mathbf{t}_i$ is the per-partition average [9]. More precisely, we map each $\mathbf{t}_i \in \mathbb{R}^D$ into $\hat{\mathbf{t}}_i \in \mathbb{R}^K$ defined by

$$\hat{t}_i(k) := \frac{1}{D_k} \sum_{d \in \mathcal{D}_k} t_i(d), \quad \text{for } 1 \leq k \leq K.$$

Assume we want to index $\mathbf{t}_i$ using any L^p norm. For $1 \leq p \leq \infty$, we define the lower-bounding norm $\|\cdot\|_{lb(p)}$ on the low-dimensional representations $\hat{\mathbf{t}}_i$ as

$$\|\hat{\mathbf{t}}_i\|_{lb(p)} := \left(\sum_{k=1}^{K} D_k \cdot |\hat{t}_i(k)|^p\right)^{\frac{1}{p}}, \text{ if } p \neq \infty, \text{ or } \|\hat{\mathbf{t}}_i\|_{lb(\infty)} := \|\hat{\mathbf{t}}_i\|_{\infty}, \text{ if } p = \infty.$$

That $\|\cdot\|_{lb(p)}$ is a lower-bounding norm for the corresponding L^p norm on the original data $\mathbf{t}_i$ is a simple extension of theorems for equal-length partitions [9]. We index $\hat{\mathbf{t}}_i$ using an R-tree (see Figure 6(b) for a simple 2D example), which recursively groups points into bounding boxes (nodes). A range query prunes nodes based on the minimum possible distance (*mindist*) of the query points to any point contained within a node. NN queries are processed by depth-first traversal and a priority queue, again using *mindist*. Since, $\|\hat{\mathbf{t}}_i\|_{lb(p)} \leq \|\mathbf{t}_i\|_p$, computing *mindist* using $\|\hat{\mathbf{t}}_i\|_{lb(p)}$ guarantees no false dismissals. We chose the partitioning $(\mathcal{D}, \mathcal{B})$ so as to make the segments as smooth as possible, therefore we expect both the node volumes to be small. Furthermore, it is precisely the smoothness that makes per-segment averages good summaries and $\|\hat{\mathbf{t}}_i\|_{lb(p)}$ a good approximation of $\|\mathbf{t}_i\|_p$.

5 Experiments

5.1 Example for Image Data

We depict with a running example the usefulness of the dimension reordering techniques for indexing and visualization. We utilize portions of the `HHRECO`[4] symbol recognition database, which consists of approximately 8000 shapes signed by 19 users. The user strokes are rendered on screen and treated as images (200×150). Since it would be unrealistic to treat each image as 200-by-150 dimensional point we perform a simple compaction of the image features as follows: by applying a $k \times m$ grid on the image, we record only $k \times m$ values which capture the number of pixels falling into each bucket. Using a 5×5 grid and starting from the top left image bucket we follow a *meander* ordering (Figure 7) and transform each image into a 25-dimensional point. The exact bucket ordering technique is of little importance, since the dimensions are going to be reordered again by our technique (therefore z- or diagonal ordering could be equally used).

[4] `http://www-cad.eecs.berkeley.edu/Respep/Research/hhreco/`

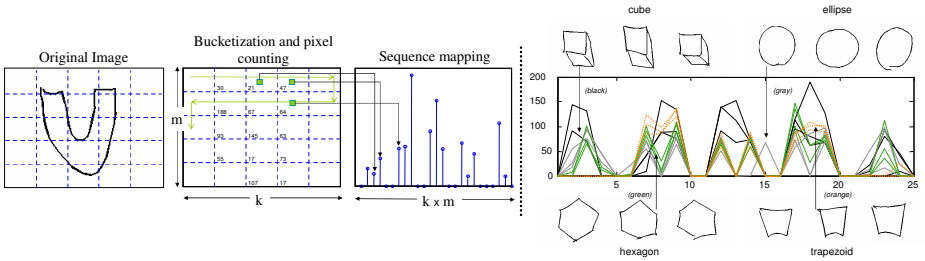


Fig. 7. *Left:* Feature extraction from image, *Right:* Mapping of features as sequences.

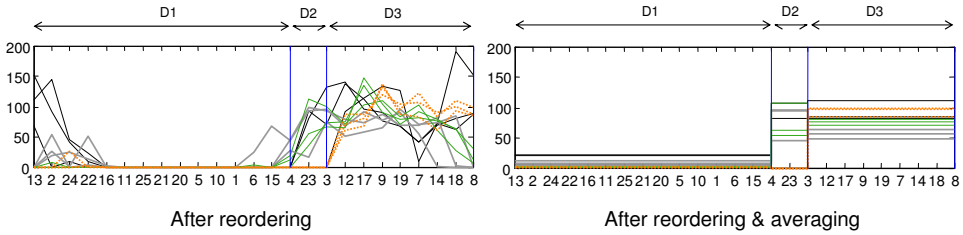


Fig. 8. *Left*: After dimension reordering, *Right*: After dimension averaging.

On the right of Figure 7 we show the originally derived 25D points for 12 images of the dataset. Figure 8 depicts the new sequences after the TSP-based reordering and also the grouping of dimensions into 3 segments. Finally the same figure also illustrates the averaging per group of projected dimensions. The new projected dimensions correspond to a *group* of the original dimensions, the correspondence of which is shown in Figure 10. Image regions corresponding to empty image space are clustered together, while image portions that carry stroke information are grouped into different segments.

Plots on projected dimensions (like Figure 8) can be very useful for summarizing and visualizing high-dimensional data. This mapping resembles the parallel coordinate technique [4]. However, our approach additionally *groups, reorders and summarizes* dimensions. When the images are projected into 2 or 3 groups of dimensions, they can also be visualized in 2D or 3D. For example, by projecting the 25D points onto 2D and placing the 12 images at their summarized projected coordinates we get the mapping of Figure 9. One can observe that relative distances are well preserved and similar-looking shapes (e.g., hexagons and circles) are projected in the vicinity of each other.

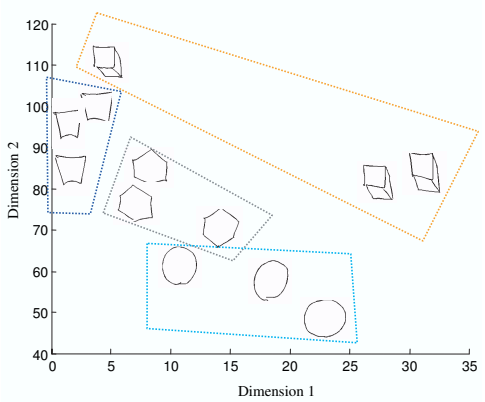


Fig. 9. 2D image mapping.

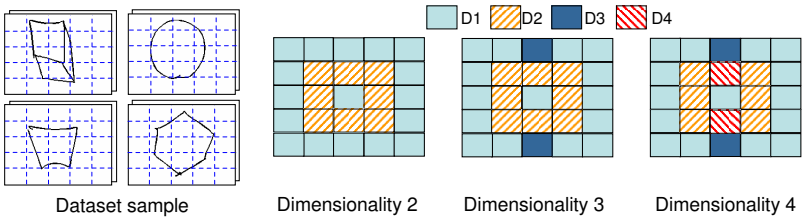


Fig. 10. Correspondence between projected dimensions and portions of the image for projected dimensionalities of 2, 3 and 4.

5.2 Application for Collaborative Filtering

We use the `MovieLens` database [5] which is utilized as a movie recommendation system. The database contains ratings for 1682 movies from 943 users. We sample a smaller portion of the database, containing all the ratings for 250 random movies and we apply our dimension ($\equiv$ movies) reordering technique. Indicative of the effective reordering is the measurement of the global smoothness, which is improved, since the cost function C that we are optimizing is minimized by a factor of 6.2. We also observed very meaningful groups of movies in the projected dimensions. For example, one grouping contains action blockbuster movies like "Indiana Jones", "Empire Strikes Back" and "Terminator", while another contains more action thrillers like "Conspiracy Theory" or "The Game".

5.3 Indexing with R-trees

Now we quantify the performance gains of our reordering and dimension grouping techniques on indexing structures (and specifically on R-trees). For this experiment we use all the images of the `HHRECO` database, but we hold out 50 random images for querying purposes. Images are converted to high-dimensional points (as discussed before), using 9, 16, 36 and 64-dimensional features. These high-dimensional features are reduced down to 3, 4, 5, 6 and 8 dimensions using the proposed methodology. The original high-dimensional data are indexed in an R-tree and their low-dimensional counterparts are also indexed in R-trees using the modified *mindist* function as discussed in Section 4.

For each method we record the amount of retrieved high-dimensional data, i.e., how many leaf records are accessed. Figure 11 displays the results normalized by the total number of data. The R-tree on the original data exhibits very little pruning power which was expected, since it operates at high dimensionality. The new R-trees operating on the grouped dimensions exhibit much higher efficiency. Notice that for 9D original dimensionality we can improve the search performance by 78% in the best case, which happens for 6 grouped dimensions. For 16D data a projected group dimensionality of 8 is the one that gives the best results, which is 62% better than the pruning power of the original R-tree. For even higher data dimensionalities, the gain from the dimension grouping diminishes slowly but one should bear in mind that the original R-tree already fetches approximately all of the data for dimensionalities higher than 16. An interesting research direction for future work would be to examine the possibility

[5] http://www.grouplens.org/

of a connection between the projected group dimensionality at which the R-tree operates most efficiently and the intrinsic data dimensionality. Realization of such a connection can lead to more effective design of indexing techniques.

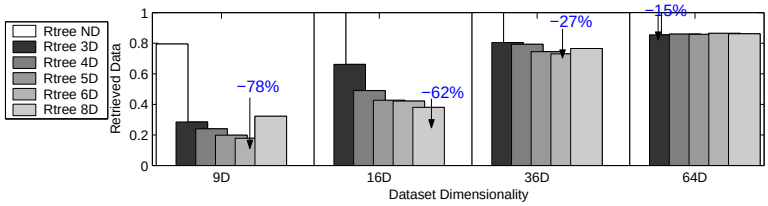


Fig. 11. Savings from using our projected grouping in conjunction with an R-tree. Data at various dimensionalities (x-axis) are projected down to 3, 4, 5, 6 and 8 dimensions.

Concluding, the indexing experiments have demonstrated that our methods can effectively enhance the pruning power of indexing techniques. We should also emphasize that essentially we have only *reorganized* and *packetized* differently the data dimensions, but we have not modified in the least in inner-workings or the structure of the R-tree index. Additionally, since there is a direct mapping between the grouped and original dimensions our technique has the additional benefit of enhanced interpretability of the results.

In the future, we plan to investigate additional ways of further improving the index performance on the new data representation. For example, in this work the grouped dimensions include all the original dimensions. However, some dimension groups are less important than others and therefore do not have to be indexed, leading to further reduction in the data dimensionality. One can see this as an analog of the largest principal components. Furthermore, such an addition to out methodology will allow for an indirect comparison with PCA, which is something that we also intend to explore in the immediate future.

6 Related work

Traditional clustering approaches, such as K-means, K-medoids or hierarchical clustering focus on finding groups of similar values and not on finding a smooth ordering, which is the main target of this work. In the mutually related fields of co-clustering, bi-clustering, subspace clustering [1, 8] (for a detailed review, see [7]) and graph partitioning [6], the problem of finding pattern similarities has been explored. Among those, pCluster [8] also tries to minimize pairwise differences, both among dimensions as well as among tuples. In general, all of these approaches focus on clustering both rows and columns and treat them symmetrically. In contrast, we assume an asymmetry ($N \gg D$) which makes the solution quite different. Most of these approaches are not suitable for large-scale databases with millions of tuples.

Mamoulis et al. [2] propose a vertical partitioning scheme for nearest neighbor query processing, which considers columns in order of decreasing variance. As pointed out before, our cost objective is not related to the per-column variance. More importantly, [2] does not provide any grouping of the dimensions, and

hence is not suitable for visualization or indexing. Our dimension summarization technique bares resemblances to the *piecewice aggregate approximation (PAA)* and *segment means* [9]; however our scheme is more general, in that, it allows segments of unequal size. Additionally those techniques perform no reordering, since they are predicated on the smoothness assumption of time-series data.

In the area of high-dimensional visualization, Fastmap [3] is a popular and fast method for dimensionality reduction and visualization. Nonetheless, it does not provide any bounds on the distance in the low-dimensional space, and therefore cannot guarantee the *no false dismissals* claim, that is provided by our lower-bounding criterion. Finally, our data representation can be seen as an extension of the *parallel coordinates* method [4]. Our technique enriches the previous model, making visualizations more coherent and useful, not only because it provides a much smoother representation, but because it also performs the additional steps of dimension grouping and summarization.

7 Conclusion

We presented RIVA, a new methodology for indexing and visualizing high-dimensional data. By expressing the data in a parallel coordinate system we attempt to discover a dimension ordering that will provide a globally smooth data representation. Such a data representation is expected to minimize data overlap and therefore enhance generic index performance as well as data visualization. We solve the dimension reordering problem by recasting it as an instance of the well-studied TSP problem. Our results indicate that R-tree performance can reap significant benefits from this dimension reorganization.

8 References

[1] C. C. Aggarwal, J. Han, J. Yang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *VLDB*, 2004.

[2] A. P. de Vries, N. Mamoulis, N. Nes, and M. L. Kersten. Efficient k-NN search on vertically decomposed data. In *SIGMOD*, 2002.

[3] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD*, 1995.

[4] A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multidimensional geometry. In *IEEE Visualization '90*, 1990.

[5] D. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *VLDB*, 2004.

[6] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *SC98*, 1998.

[7] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE Trans. Comp. Biol. and Bioinf.*, 1(1), 2004.

[8] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD*, 2002.

[9] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary $\mathcal{L}_p$ norms. In *VLDB*, 2000.