2013 IEEE 13th International Conference on Data Mining

Efficient Visualization of Large-scale Data Tables through Reordering and Entropy Minimization

Nemanja Djuric
Dept. of Computer and Information Sciences
Temple University, Philadelphia, USA
e-mail: nemanja@temple.edu

Slobodan Vucetic
Dept. of Computer and Information Sciences
Temple University, Philadelphia, USA
e-mail: vucetic@temple.edu

***Abstract*—Visualization of data tables with n examples and m columns using heatmaps provides a holistic view of the original data. As there are $n!$ ways to order rows and $m!$ ways to order columns, and data tables are typically ordered without regard to visual inspection, heatmaps of the original data tables often appear as noisy images. However, if rows and columns of a data table are ordered such that similar rows and similar columns are grouped together, a heatmap may provide a deep insight into the underlying data distribution. We propose an information-theoretic approach to produce a well-ordered data table. In particular, we search for ordering that minimizes entropy of residuals of predictive coding applied on the ordered data table. This formalization leads to a novel ordering procedure, EM-ordering, that can be applied separately on rows and columns. For ordering of rows, EM-ordering repeats until convergence the steps of (1) rescaling columns and (2) solving a Traveling Salesman Problem (TSP) where rows are treated as cities. To allow fast ordering of large data tables, we propose an efficient TSP heuristic with modest $\mathcal{O}\big(n\log(n)\big)$ time complexity. When compared to the existing state-of-the-art reordering approaches, we show that the method often provides heatmaps of higher visual quality, while being significantly more scalable. Moreover, analysis of real-world traffic and financial data sets using the proposed method, which allowed us to readily gain deeper insights about the data, further confirmed that EM-ordering can be a valuable tool for visual exploration of large-scale data sets.**

I. Introduction

Data visualization has a long history in scientific research [1] as it allows researchers to get a better insight into data they are studying. Visualization is used for exploratory analysis prior to application of statistical methods, it can also be used as a confirmatory tool to disprove or confirm hypotheses, while sometimes visual presentation is an ultimate goal [2]. However, despite its long history and significant advantages of visual analysis of data [3], there still remains a need for further development of visualization methods. This is particularly evident when working with large-scale, high-dimensional data, where commonly used visualization tools are either too simplistic to gain a deeper insight into the data properties (e.g., histograms, scatter plots, pie and bar charts), or are too cumbersome and computationally costly in large-scale setting, such as parallel coordinates [4], [5], correlation matrix plots [6], and biplots and star plots [3]. Inadequacy of standard tools has been recognized in a number of recent papers, as summarized in the statement from [7]: *Data in high dimension are difficult to visualize and understand. This has always been the case and is even more apparent now with the availability of large high-dimensional datasets and the need to make sense of them.*

In this paper, we focus on visualizing data sets that can be represented as an $n \times m$ data table, where rows represent n examples and columns represent m features. Standard data exploration tools such as histograms and scatter plots provide only a basic understanding of the data; histogram is a tool for understanding distributions of each feature, while scatter plot is a tool for understanding correlations between pairs of features. A more advanced visualization approach is low-dimensional data projection, where examples are projected into a two-dimensional subspace and visualized using a scatter plot, such as Principal Component Analysis (PCA), Locally Linear Embedding (LLE) [8], or Stochastic Neighborhood Embedding (SNE) [9]. However, projecting examples into a 2-D subspace and visualizing them using a scatter plot often implies a significant loss of information. Moreover, while the resulting 2-D or 3-D scatter plots can provide insight into an underlying manifold structure, they may be difficult to interpret and provide actionable knowledge. This is evident when examining related publications that typically use non-linear projection methods to project a set of images (e.g., faces, digits) to a 2-D scatter plot, and then plot the original image next to the corresponding projected point to illustrate the quality of visualization. However, practical problem is that there are only several types of data sets where the projected examples can be conveniently annotated in a lower-dimensional plot.

An alternative to showing two- and three-dimensional scatter plots of the projected data is to plot the original data. By observing that a data set can be represented as a two-dimensional matrix, it becomes evident that is could be plotted as a heatmap (e.g., using `imagesc` command in Matlab and Octave). Since examples and features in a typical data set are sorted in an arbitrary order (e.g., randomly, or by example or feature ID), heatmap of the original data might not be informative. There are two possible alternatives for improving the heatmap. One is to perform clustering (e.g., k-means clustering) and sort all examples based on which cluster they are assigned to. However, the outcome greatly depends on the chosen number of clusters, and could result in artifacts where it might appear that there are clear clusters even when this is not the case. More appropriate strategy for plotting data heatmaps is to first reorder its rows (columns), such that similar rows (columns) are placed next to each other [10], [11], [12].

There are many possible approaches for ordering of data tables. One is to project examples onto the largest principal component obtained by PCA or to a principal curve obtained by LLE, and to order the examples by traversing the line or

1550-4786/13 $31.00 © 2013 IEEE
DOI 10.1109/ICDM.2013.63

the curve. However, ordering is not an explicit objective of either PCA or LLE but only a byproduct of a manifold search, and may result in lower-quality visualization. An alternative, very popular in gene expression analysis, is to perform hierarchical clustering and to order examples by traversing leaves of the binary tree [13]. However, the resulting algorithm is computationally expensive and can be applied only to data sets with several thousand examples. Moreover, there are 2^{n-1} ways to order the resulting hierarchical tree, which may open a costly optimization problem [14]. Another interesting approach presented in [15] is ordering based on spectral clustering. Similarly to hierarchical clustering approaches and unlike the method proposed in this paper, in large-scale setting spectral analysis becomes time- and memory-intensive, and the algorithm may also give suboptimal results when the data consists of several clusters that are not well separated.

Data table reordering can also be seen as the Traveling Salesman Problem (TSP) [16], where examples represent cities, and the task is to find a path through all the cities such that the traversal cost is minimized. While TSP is an NP-complete problem that requires exponential computation time, there are efficient heuristics that in practice give high-quality tours. The Lin-Kernighan (LK) method [17] has been widely accepted as the method providing the best trade-off between tour quality and computational speed, scaling as $\mathcal{O}(n^{2.2})$; as such, it is applicable only to moderately-sized data sets. Moreover, treating reordering directly as TSP carries a strong assumption that the features were properly scaled from the perspective of visual quality of heatmaps.

We propose a novel ordering method that addresses shortcomings of the existing methods. Our main contributions are:

- Ordering is formalized as finding a permutation of rows (or columns) that results in a maximally compressible data set, as defined by the entropy of the residuals of predictive coding.
- The problem is solved by an Expectation-Maximization (EM)-like algorithm, which alternatively solves a TSP and reweights features based on the quality of the resulting tour.
- A fast $\mathcal{O}\big(n\log(n)\big)$ TSP solver is proposed, called the TSP-means, that finds tours with lengths comparable to those found by the LK algorithm. It is based on a construction of a binary tree by recursive use of k-means (with $k = 2$) and subsequent reordering of the tree nodes by the LK algorithm.

II. Background

In this section we describe the works and ideas that led to the proposed visualization method. We first introduce the existing approaches for visualization of high-dimensional data, and then present matrix reordering and data seriation techniques. Lastly, we give an overview of the TSP and the existing methods for solving this classical combinatorial problem.

A. Data visualization

Visualization of data has been an integral part of scientific research from the earliest times, with visual representations of data appearing in scientific literature from as early as the 10^{th} century [1]. A great number of approaches for data visualization has been introduced since (see [18], [19]), with visualization methods most commonly used in our everyday lives, such as histograms and pie charts often encountered in newspaper and weather reports, being in use for more than a century in a nearly unchanged form [1], [20], [21]. However, recent technological advances and emergence of large-scale data sets have clearly indicated limitation of the existing methods in this new setting [2], [7], and there remains a clear need for the development of novel visualization approaches.

Visualization of high-dimensional data is of particular interest [7], and this problem has received significant attention in the visualization community. Often explored direction is finding lower-dimensional representation of the data, which could then be more easily visualized using the standard visualization tools. In [22] and [23], the authors propose methods that explore interactions between examples in subspaces of the original high-dimensional space, and plot these lower-dimensional representations in a form of similarity matrices or scatter plots in order to gain better understanding of the data. However, the methods become intractable as number of examples and dimensions grows, and may not be suitable for large-scale visualization tasks. Instead of using subspace search, another idea is to compute more involved projections of the data into 2- or 3-D spaces. This approach includes PCA, where examples are projected along the directions describing most of the variance, and non-linear projections such as LLE [8], SNE and its extension t-SNE [9], [24], Self-Organizing Maps (SOM) [25], Isomap [26] or Laplacian eigenmaps [27], which attempt to project examples to a lower-dimensional, non-linear manifold. However, lower-dimensional projection methods in most cases imply a significant loss of information, and the resulting plots may also be difficult to interpret by non-experts for whom the visualization results are often intended.

To address this issue, an interesting approach is to represent examples in their original, high-dimensional space. A very popular technique implementing this idea are parallel coordinates [4], [5], which have been used for data visualization for more than a century [1]. However, although parallel coordinates can be used to quickly discover trends in moderate-sized data sets, they become cluttered when number of examples or dimensions becomes large [28], [29], [30], thus significantly limiting the quality of visualization. On the other hand, an alternative visualization tool are heatmaps, with very long and rich history [20], [21]. In contrast to parallel coordinates, heatmaps do not suffer from the deficiencies related to extreme data sizes, and can be used in large-scale setting to provide a holistic view of the data. We use this insight and propose a scalable algorithm for generating high-quality, large-scale heatmaps, obtained by data preprocessing through reordering.

B. Data reordering

Data reordering or seriation is an important step in exploratory data analysis. This family of unsupervised methods is based on the following observation: assuming that a data set is represented in a form of a two-dimensional reorderable matrix, any permutation of its columns or rows does not lead to loss of information [11]. Therefore, by permuting rows (columns) so that similar rows (columns) are close, followed by visualization of the modified data matrix, we can reveal unknown regularities and patterns in the data without modifying the data. Data ordering has deep roots in a number of disciplines in social studies (e.g., archeology [31], anthropology [32];

for an excellent overview see [33] and references therein). Beyond social sciences, data ordering has been popular in gene expression data analysis in bioinformatics [13], [14] and analysis of geographical data [34]. It is also important in bandwidth minimization [35] and data compression [36], [37].

We describe in more detail several popular methods for data ordering. As baseline methods we can consider LLE and PCA, two popular low-dimensional projection algorithms. One-dimensional projection by either PCA or LLE effectively induces a linear ordering in the new 1-D space, and can be used to reorder rows and columns of the data matrix. LLE method first finds k nearest neighbors in the original high-dimensional space for each example, then attempts to project the data to lower-dimensional space while keeping the relationships between neighbors the same. Due to the fact that all distances between examples need to be computed, the time complexity of the algorithm amounts to $\mathcal{O}(n^2)$. Regarding PCA, we can use the first principal component and project the data onto it. As we only need to compute the first principal component (found in $\mathcal{O}(n)$ [38]), this algorithm is very fast, and projection of the data to the first principal component and subsequent sorting of the projected values result in modest $\mathcal{O}\big(n\log(n)\big)$ time complexity. We can also consider ordering based on spectral clustering (SC) [15], which can be seen as a low-dimensional, non-linear projection method. In their work, the authors show that linear ordering of examples can be found by computing the second largest eigenvector of the normalized similarity matrix. However, similarly to LLE, time and space complexity of $\mathcal{O}(n^2)$ render the method infeasible in large-scale setting.

Two approaches popularized in bioinformatics are hierarchical clustering (HC) [13] and hierarchical clustering with optimal leaf ordering (HC-olo) [14]. Hierarchical clustering is a bottom-up method, which starts by clustering two most similar examples and represents this new cluster with its centroid. Examples and centroids are repeatedly grouped together until all examples belong to a single, root cluster. The method finds binary tree representation of the data, resulting in $\mathcal{O}(n^2)$ time complexity as distances between all examples need to be calculated. In order to find ordering of examples, we simply read leaves of the tree from left to right. However, there are 2^{n-1} linear orderings of tree nodes that obey the obtained tree structure (i.e., we can flip each of $n-1$ internal nodes and still have the same tree structure). To solve this issue, in [14] the authors present a dynamic programming approach to find the optimal leaf ordering for a given tree structure in $\mathcal{O}(n^3)$ time, which makes the algorithm intractable for larger data sets.

C. Traveling salesman problem (TSP)

Traveling Salesman Problem is a classical problem in computer science. The problem can be described as follows: given n cities, along with non-negative costs of traveling from the i^{th} city to the j^{th} city $d(i,j), i,j \in \{1,2,...,n\}$, find a shortest path such that each city is visited exactly once and the path completes in the starting city. More formally, letting π be permutation (or ordering) of n cities, the task is to find optimal permutation π^* so that the total tour length is minimized,

$$\pi^* = \underset{\pi\in\Pi_n}{\arg\min}\left(d\big(\pi(n),\pi(1)\big) + \sum_{i=2}^{n} d\big(\pi(i-1),\pi(i)\big)\right), \tag{1}$$

where Π_n is the set of all permutations of the first n integers, and $\pi(i)$ denotes the i^{th} city to be visited. The TSP is NP-complete [39], thus very difficult to solve optimally.

Due to the NP-completeness of the problem, TSPs were historically very hard to solve with limited computational resources. One of the first large TSP problems solved to optimality involved only 49 cities, and the solution was described in 1954 in [40]. Interestingly, the authors solved the problem by manually applying ideas that later led to the cutting-plane algorithm [41]. Several polynomial-time heuristic methods with upper bounds on performance have been proposed since to approximately solve the TSP, including nearest neighbor, nearest insertion, furthest insertion [42], Christofides heuristic [43], and LK heuristic (see [44] for a comprehensive overview of methods). Currently, the largest TSP instance solved to optimality comprises nearly $86{,}000$ cities [45].

One of the most powerful heuristics for finding optimal or near-optimal solutions to TSP is the LK method, having $\mathcal{O}(n^{2.2})$ time complexity. The method introduces a variable λ-opt move (where $\lambda \geq 2$) to reach a better tour, meaning that at each iteration we search for increasing λ number of links on the current tour that could be broken and replaced by the same number of links currently not on the tour. The λ-opt move is performed if the resulting, modified tour has lower cost than the current solution. The method starts with a random tour, and then iteratively applies λ-opt moves, until no such move leads to a better solution (it is then said that the current tour is λ-optimal). An efficient implementation of LK heuristic is presented in [46], which achieved the best results on all known large-scale TSP problems.

An interesting application of TSP solvers is in matrix reordering and clustering, where each data example is considered a city. Interestingly, one of the first matrix reordering techniques, the Bond Energy Algorithm (BEA), is in fact a simple nearest insertion TSP heuristic [47]. In [16], the authors propose adding several "dummy" cities which have distances equal to 0 to all other cities. In this way, after computing the shortest tour through all cities, the "dummy" cities act as boundaries between different clusters. However, as discussed in [48] and as shown in the experimental section of this paper, directly applying TSP to the whole data set can lead to ordering that is very sensitive to noise inherent in the data set, and consequently to poor data visualization. In [49] the authors propose a TSP-based biclustering method, which first finds clusters across rows, followed by clustering across columns of a data matrix to obtain meaningful biclusters. As the proposed methods apply TSP solvers directly on the whole data set, they are not scalable to large data sets due to super-quadratic time complexity of the best available TSP solvers. In contrast to the existing methods, we present an effective algorithm with time requirement of only $\mathcal{O}(n\log(n))$, allowing for high-quality, scalable reordering and visualization of large data matrices.

III. Methodology

An intuitive goal of ordering is to find permutation of rows so that similar examples are grouped together. We propose a principled approach for ordering that finds permutation of rows producing a maximally compressible data set. We will explain how to order rows, while noting that columns can be ordered using the same procedure on the transposed data table.

A. Differential Predictive Coding (DPC)

Let us assume a data set D is given in a form of an $n \times m$ data table, $D = [x_{ij}]_{i=1,\ldots,n,j=1,\ldots,m}$, where the i^{th} row vector $\mathbf{x}_i = [x_{i1}, \ldots, x_{im}]$ is the i^{th} example having m numeric features. DPC replaces the i^{th} example $\mathbf{x}_i$ with its difference from the previous example, $\boldsymbol{\varepsilon}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$, where $\boldsymbol{\varepsilon}_i$ is called the DPC residual. Therefore, DPC transforms the data table $D = [\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}}, \ldots, \mathbf{x}_n^{\mathrm{T}}]^{\mathrm{T}}$ into $D_{DPC} = [\mathbf{x}_1^{\mathrm{T}}, \boldsymbol{\varepsilon}_2^{\mathrm{T}}, \ldots, \boldsymbol{\varepsilon}_n^{\mathrm{T}}]^{\mathrm{T}}$ without the loss of information, as the original data set D can be reconstructed from D_{DPC}.

If the data table D is ordered such that similar rows are placed next to each other, DPC residuals $\boldsymbol{\varepsilon}$ would be smaller than the original examples $\mathbf{x}_i$. As a result, the entropy of rows in D_{DPC} would be smaller than the entropy of rows in D, indicating that D_{DPC} is more compressible than D. The entropy of the original examples is defined as $H_D(\mathbf{x}) = \mathbb{E}[-\log(\mathbb{P}_{\mathcal{X}}(\mathbf{x}))]$, where $\mathbb{P}_{\mathcal{X}}(\mathbf{x})$ is the probability density of vectors $\mathbf{x}$, and entropy of DPC residuals is defined as $H_{DPC}(\boldsymbol{\varepsilon}) = \mathbb{E}[-\log(\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon}))]$, where $\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon})$ is a probability density of vectors $\boldsymbol{\varepsilon}$. Thus, small entropy $H_{DPC}(\boldsymbol{\varepsilon})$ implies that DPC residuals are small, which in turn implies that D is a well-ordered data set. As a result, entropy of the DPC residuals is a good measure of ordering quality. We note that $H_{DPC}(\boldsymbol{\varepsilon})$ can be estimated as

$$H_{DPC}(\boldsymbol{\varepsilon}) = -\frac{1}{n-1}\sum_{i=2}^{n} \log \mathbb{P}_{\mathcal{E}}(\mathbf{x}_i - \mathbf{x}_{i-1}). \tag{2}$$

B. Relationship between entropy minimization and ordering

In data mining, data sets can often be considered as arbitrarily ordered collections of examples and features. As a consequence, any permutation π of rows from $D = [\mathbf{x}_i]_{i=1,\ldots,n}$, resulting in $D_\pi = [\mathbf{x}_{\pi(i)}]_{i=1,\ldots,n}$, does not lead to loss of information. We propose that the optimal permutation π^* is the one that results in minimization of entropy of DPC residuals,

$$\pi^* = \underset{\pi \in \Pi_n}{\arg\min}\, H^{\pi}_{DPC}, \tag{3}$$

where we used superscript π to denote the specific permutation of rows of the data table D.

We observed that when applying DPC on a number of well-ordered data sets with numerical features $\mathbb{P}_{\mathcal{E}}$ often resembles multivariate Gaussian or Laplacian distribution with diagonal covariance matrix. Let us first consider the case when $\mathbb{P}_{\mathcal{E}}$ is a multivariate Gaussian distribution,

$$\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon}) = (2\cdot\pi)^{-m/2}|\Sigma|^{-1/2}\cdot\exp(-0.5\cdot\boldsymbol{\varepsilon}^{\mathrm{T}}\cdot\Sigma^{-1}\cdot\boldsymbol{\varepsilon}), \tag{4}$$

where Σ is a diagonal covariance matrix with the j^{th} element on a diagonal equal to the variance σ_j^2 of the DPC residuals of the j^{th} feature. $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$ could then be expressed as

$$H^{\pi}_{DPC}(\boldsymbol{\varepsilon}) = \frac{n}{2(n-1)}\left(m\cdot\log(2\pi) + \sum_{j=1}^{m}\log\sigma_j\right) + \frac{1}{2(n-1)}\sum_{i=2}^{n}\sum_{j=1}^{m}\frac{(x_{\pi(i),j} - x_{\pi(i-1),j})^2}{\sigma_j^2}. \tag{5}$$

When $\mathbb{P}_{\mathcal{E}}$ is modeled as a Laplacian distribution, and assuming independence of elements of $\boldsymbol{\varepsilon}$, $\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon})$ is equal to

$$\mathbb{P}_{\mathcal{E}}(\boldsymbol{\varepsilon}) = \prod_{j=1}^{m}\frac{1}{2b_j}\exp(-\frac{|\boldsymbol{\varepsilon}_j|}{b_j}), \tag{6}$$

where $b_j^2 = \sigma_j^2/2$. The corresponding $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$ is similar to (5), with the main difference being that $|\sigma_j|$ is used instead of σ_j^2 in the third term of (5).

Upon modeling $\mathbb{P}_{\mathcal{E}}$ as Gaussian or Laplacian distribution, and observing that this results in introduction of m new parameters $\{\sigma_j\}_{j=1,\ldots,m}$, we can restate (3) as

$$(\pi^*, \{\sigma_j^*\}_{j=1,\ldots,m}) = \underset{\pi,\{\sigma_j^*\}_{j=1,\ldots,m}}{\arg\min}\, H^{\pi}_{DPC}(\boldsymbol{\varepsilon}). \tag{7}$$

Solving (7) requires finding the best ordering and the best estimate of variance of DPC residuals for each feature.

C. Reordering for entropy minimization

We propose to solve (7) in an iterative manner similar to the EM algorithm. The method is given as Algorithm 1. In the M-step (line 2), by assuming that values of σ_j are known, the problem reduces to finding ordering π that minimizes the last term from (5), which is equivalent to solving the TSP on examples whose features are downscaled using $\{\sigma_j\}_{j=1,\ldots,m}$. Given the current ordering π, the goal of the E-step (line 3) is to find $\{\sigma_j\}_{j=1,\ldots,m}$ that minimizes $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$. It is evident that the E-step is equivalent to finding σ_j using the maximum likelihood approach, where σ_j is found as

$$\sigma_j^2 = \frac{1}{n-1}\sum_{i=2}^{n}(x_{\pi(i),j} - x_{\pi(i-1),j})^2. \tag{8}$$

Algorithm 1 converges to a local minimum, since both E- and M-steps lead to decrease in $H^{\pi}_{DPC}(\boldsymbol{\varepsilon})$. As the algorithm resembles the EM and is based on an information-theoretic principle of Entropy Minimization, we call it the EM-Ordering.

Note that successful ordering will result in small σ_j for features that are correlated to others, and large σ_j for noisy or uncorrelated features. Intuitively, if the j^{th} feature cannot be ordered well during the procedure, its DPC entropy, and thus σ_j will be increased. As a result, its importance will be reduced due to larger downscaling in (5).

D. Feature scaling

We observe that the proposed algorithm allows for cases when for some features and for some orderings it holds that $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$, where $H_D(\mathbf{x}_j)$ and $H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$ are entropies of the j^{th} feature and of its DPC residuals, respectively. If this is the case, it might be preferable to ignore the j^{th} feature during ordering. Considering this observation, we propose two strategies:

1. Hard feature scaling. In this case, features for which holds that $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$ are scaled down to zero by introducing $\sigma_j \to \infty$, in order to prevent their adverse influence on the overall entropy minimization.

2. Soft feature scaling. In this case, no action is being taken for features where $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\boldsymbol{\varepsilon}_j)$.

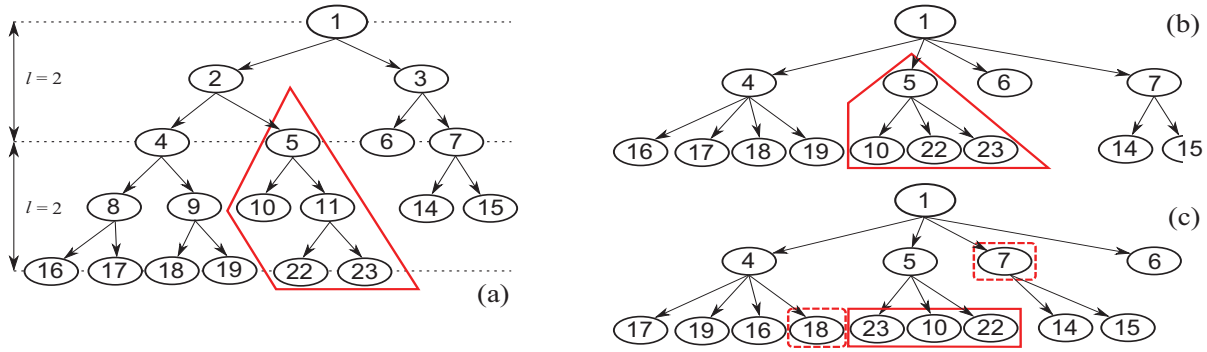

Fig. 1. (a) Binary tree after recursive k-means, with $k = 2$; (b) Resulting 2^l-ary tree ($l = 2$), obtained after line 3 in Algorithm 2; (c) TSP defined on node 5 is solved on its children, together with left and right neighbor nodes 18 and 7, see Algorithm 3

Algorithm 1 EM-ordering

Inputs: data set D; initial guess for $\{\sigma_j\}_{j=1,\ldots,m}$
Output: ordered set D; learned $\{\sigma_j\}_{j=1,\ldots,m}$

1. **repeat** until convergence
2. **run** TSP solver for current σ_j to find π
3. **calculate** σ_j for current ordering of D

Algorithm 2 Generation of 2^l-ary tree T^l

Inputs: binary tree T; subtree depth parameter l
Output: 2^l-ary tree T^l

1. **extend** leaf nodes of T at levels $\big((i-1)\cdot l+1\big)$ through $(i\cdot l-1)$ to level $i\cdot l, i>0$
2. **select** nodes of T at levels $i\cdot l, i\geq 0$ for inclusion in tree T^l
3. **connect** nodes in T^l originating from level $i\cdot l$ in T, $i>0$, to their predecessor at level $(i-1)\cdot l$ from T
4. **add** children to every leaf of T^l, where the children are individual examples in that leaf's cluster

Hard scaling results in a removal of the features that cannot be successfully ordered and, as such, can be considered a feature selection algorithm. However, to prevent removing features that at a given iteration just barely satisfy the condition $H_D(\mathbf{x}_j) < H^{\pi}_{DPC}(\varepsilon_j)$, but could be successfully ordered in future iterations of Alg. 1, we can use the following criterion:

3. Hard feature scaling with tolerance. The j^{th} feature is removed if, for some $\alpha > 1$, $H^{\pi}_{DPC}(\varepsilon_j) > \alpha H_D(\mathbf{x}_j)$.

E. TSP-means algorithm

In this section, we propose a TSP solver used in M-step of Algorithm 1, called TSP-means. By combining data clustering and efficiently solving a number of small-scale TSPs defined on the cluster centroids, we obtain a highly scalable algorithm. In addition, as will be discussed later, by its design TSP-means often results in a more informative visualization than when LK is directly used on all examples.

The solver, summarized in Algorithm 3, begins by recursively applying k-means clustering with $k = 2$ (line 1), to create a binary-tree T representation of the data set D, as shown in Figure 1(a). The root of the tree corresponds to the whole data set and is represented by its centroid. Internal nodes correspond to clusters found by running k-means on their parents, and are represented by the cluster centroids. The k-means on a node is not performed if a node contains less than or exactly 2^l examples, where l is a user-defined parameter.

Algorithm 3 TSP-means

Inputs: data set D; subtree depth parameter l
Output: ordered list $\mathcal{L}$

1. **create** binary tree T by recursively splitting D
2. **run** Algorithm 2 on T to generate T^l
3. **set** $\mathcal{L} \leftarrow$ root r of T^l
4. **while** (not all leaf nodes of T^l in $\mathcal{L}$)
5. **for each** z in L in left-to-right order
6. **if** (z has children)
7. **solve** local TSP defined on children of z and immediate neighbors of z in the list $\mathcal{L}$
8. **replace** z in $\mathcal{L}$ by its children, in order given by the TSP solution

In the next step, as formalized in Algorithm 2 and illustrated in Figure 1(b), we transform binary tree T into 2^l-ary tree T^l by keeping only nodes at every l^{th} tree level, starting with a root node. Each node at level $(i\cdot l)$, $i>0$, becomes a child of its predecessor at $((i-1)\cdot l)^{\text{th}}$ level (e.g., nodes 22 and 23 become children of node 5). In addition, leafs at any level in the tree T also become leafs in the tree T^l. For example, node 10 becomes a child of node 5, as shown in Figure 1(b). Note that leafs of the 2^l-ary tree T^l represent clusters with no more than 2^l examples, and are the same as leafs of T. Finally, we add another tree level to T^l by making its current leaf nodes parents of examples in their cluster. This results in the final 2^l-ary tree whose leaf nodes are the individual examples.

After the creation of 2^l-ary tree, we perform a breadth-first traversal of the tree from left to right (Alg. 3, lines 3 - 8). The main idea is to reorder the internal and leaf nodes so that similar clusters and examples are closer, resulting in a good ordering of the data set. For this purpose we create a list $\mathcal{L}$, which initially holds only the root of the tree (Alg. 3, line 3). Then, we visit nodes in the list $\mathcal{L}$ sequentially from left to right and solve a local TSP defined on centroids of children of the current node in $\mathcal{L}$, giving us an ordering where similar children are close (Alg. 3, lines 4 - 7). Before moving on to the next

TABLE I. COMPLEXITIES OF THE REORDERING ALGORITHMS

Algorithm	Time	Space
PCA	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n)$
LLE	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
SC	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
HC	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
HC-olo	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
LK	$\mathcal{O}(n^{2.2})$	$\mathcal{O}(n)$
TSP-means	$\mathcal{O}\big(n\log(n)\big)$	$\mathcal{O}(n)$

node of the list $\mathcal{L}$, we replace the current node in the list $\mathcal{L}$ with its children reordered according to the TSP solution (Alg. 3, line 8). Once we reach the end of the list $\mathcal{L}$, we start again from the beginning. For example, for the tree in Figure 1(b), we initialize $\mathcal{L} = [1]$. We traverse the list from left to right, and, after solving TSP defined on children of the current node of the list (there is only node 1 at the moment), we replace node 1 in the list $\mathcal{L}$ with its children in the order given by the TSP solution, resulting in $\mathcal{L} = [4, 5, 7, 6]$, see Figure 1(c). As we reached the end of the list, we start from the beginning and the current node of $\mathcal{L}$ becomes node 4, whose children define the next TSP to be solved. The algorithm completes when $\mathcal{L}$ contains only individual examples, and ordered data set is returned in a form of the resulting list $\mathcal{L}$.

TSP tour computed only on children of the current node of $\mathcal{L}$ would result in discontinuity between tours of the neighboring nodes in $\mathcal{L}$, as each local tour would be computed irrespectively of the neighboring nodes. Therefore, to ensure that the ordering is smooth, when solving a TSP defined on children of the current node in $\mathcal{L}$ we also include its left and right neighbors from $\mathcal{L}$ if they exist. We set the distance between left and right neighbor nodes to 0, so that they are guaranteed to become neighbors on the found tour. After solving thus defined TSP, the found tour is cut at these neighbors to obtain an ordered list, and the children of the current node are reordered according to the TSP solution. For example, given $\mathcal{L} = [17, 19, 16, 18, 5, 7, 6]$, obtained after solving TSP defined by the previous current node 4, we need to solve the TSP defined by the new current node 5. The TSP being solved includes nodes $\{18, 10, 22, 23, 7\}$, and the resulting tour is $[18, 23, 10, 22, 7]$. Before moving on to node 7, we replace node 5 with its ordered children to obtain the updated list $\mathcal{L} = [17, 19, 16, 18, 23, 10, 22, 7, 6]$, see Fig. 1(c).

F. Further details

There are several important advantages that TSP-means offers over the existing algorithms. First, it has very favorable time complexity. As k-means clustering has time requirement linear in number of examples, it takes $\mathcal{O}\big(n\log(n)\big)$ time to build the binary tree T, assuming k-means results in nearly balanced clusters. After creating T^l in $\mathcal{O}(n)$ time, there are $\mathcal{O}(\frac{n}{2^l})$ nodes in T^l, assuming a nearly balanced tree T^l of expected depth close to $\log(n)$. Each non-leaf node in T^l requires solving a single TSP whose size is bounded by (2^l+2). Therefore, if LK algorithm is used for TSP, time complexity of solving each TSP is $\mathcal{O}(2^{2.2l})$. It follows that the overall time requirement of TSP-means is only $\mathcal{O}\big(2^{1.2l}n\log(n)\big)$. Summary of time and space complexities of TSP-means and the competing methods is given in Table I.

TSP-means is also amenable to parallelization. In particular, clustering of non-root nodes can be distributed over multiple processors, while solving TSPs defined on higher-level nodes can be performed concurrently with clustering of lower-level nodes. In addition, the algorithm allows for interactive visualization of large data sets. Unlike the competing algorithms, where a user can plot the results only once the algorithm completes, TSP-means provides meaningful output during the execution at each depth of the tree. It can visualize the data during runtime, by first showing coarse data ordering, and then transitioning (or "zooming in") towards finer resolutions as the algorithm descends further down the tree. In particular, as the list $\mathcal{L}$ is expanded we can show centroids currently in the list, weighted by the number of examples in the cluster that they represent. In this way, useful local and global patterns can be discovered even before completely traversing the tree T^l. Note that TSP-means can be further sped up by running k-means on a sub-sampled data for each node in the tree T instead on the whole data set, resulting in constant-time calls to k-means.

TABLE II. EVALUATION OF PERFORMANCE ON *waveform* DATA OF SIZE 10,000; LISTED LK TIME IS REQUIRED FOR SOLVING ONE TSP

l	# LK	LK time [sec]	# k-means	FOM	L
2	4,721	0.000	5,224	0.234	31,663
4	2,496	0.002	3,541	0.235	31,244
6	2,304	0.035	3,604	0.233	30,925
7	533	0.067	1,369	0.229	29,840
8	257	0.253	255	0.228	30,823
9	504	0.516	506	0.233	30,420
10	985	1.093	1,019	0.230	30,007

IV. EXPERIMENTS

In all experiments, initial values of variances $\{\sigma_j\}_{j=1,\ldots,m}$ in EM-ordering were initialized to standard deviation of features. We used hard feature scaling with tolerance of $\alpha = 1.1$, and run EM-ordering for 5 iterations. We assumed a Gaussian distribution from (4) for DPC residuals. To build a tree T, at each step of recursion we run k-means ($k = 2$) on 100 randomly sampled examples. For HC and HC-olo we used the Bioinformatics toolbox in Matlab (with average linkage). SC was implemented in Matlab (similarity matrix computed using Gaussian kernel), while codes for LLE[1] (number of neighbors was set to 15), and Lin-Kernighan[2] were found online.

It is not obvious how to measure quality of visualization. As a proxy measure we used Figure of Merit (FOM) [50] when labeled examples are available. Denoting label of the i^{th} example as $y(i)$, FOM score of ordering π is computed as

$$\text{FOM}(\pi) = \frac{1}{n-1}\sum_{i=1}^{n-1} I\Big(y\big(\pi(i)\big) \neq y\big(\pi(i+1)\big)\Big), \qquad (9)$$

where binary indicator function $I(\cdot)$ returns 1 if the argument is true, and 0 otherwise. As a result, lower values of FOM indicate higher-quality ordering π. To evaluate TSP-means, we also report tour cost L, equal to the sum of Euclidean distances between neighboring examples in the final ordering.

A. Validation of TSP-means

We first evaluated influence of parameter l in Alg. 3. The results for *waveform* data set of size $10{,}000$, for l ranging

[1] http://cs.nyu.edu/∼roweis/lle/code.html, accessed June 2013

[2] http://www.tsp.gatech.edu/concorde.html, accessed June 2013

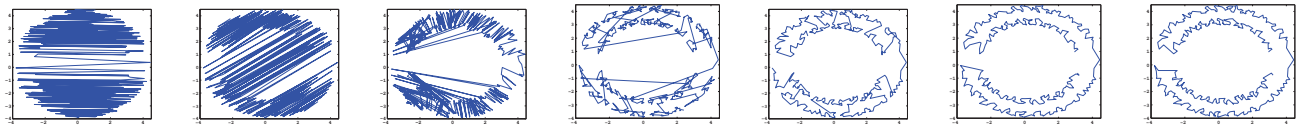

Fig. 2. Performance of ordering algorithms on *circles* data (n = 500, path length L is given in parentheses): (a) PCA (1,131.9); (b) LLE (956.3); (c) SC (432.1); (d) HC (164.5); (e) HC-olo (94.5); (f) LK (83.2); (g) TSP-means (84.5)

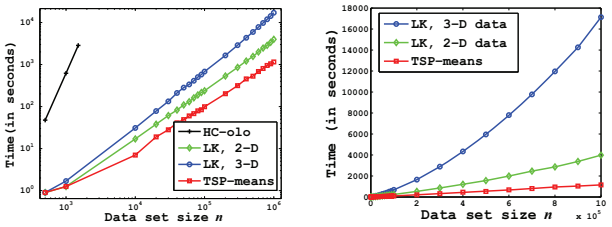

Fig. 3. Execution times on 2-D and 3-D *uniform* data set: (a) logarithmic scale; (b) linear scale

from 2 to 10, are given in Table II, where we report number of calls to LK and k-means sub-routines, time required to solve a single TSP, as well as FOM and L performance measures. As can be seen, setting l to 7 led to good results. This can be explained by the fact that depth of the tree T (after step 1 in Algorithm 3) was 14, as expected since $\lceil \log_2(10{,}000) \rceil < 14$, meaning that the depth of T^l was only 2. For $l < 7$ the depth of T^l was larger than 2, leading to more calls to LK and k-means subroutines. On the other hand, for $l > 7$ the TSP of root node of T^l had 2^l children, while TSPs of non-root nodes had less than 2^{14-l} children, thus not fully exploiting near-optimal ordering found by LK at the lower levels. Taking this result into consideration, in the remaining experiments we set $l = \lceil 0.5 \ \log_2(n) \rceil$ as a default value.

To get an insight into relative performance of the competing algorithms, we generated 2-dimensional data set called *circles*. We uniformly at random sampled half of the examples from a circle of radius 3, and the second half from the circle of radius 4. We also added small noise to all examples. We set the size of the *circles* data set to 500 for the clarity of visualization. The resulting ordering of all methods is shown in Figure 2, where neighboring examples in the final linear ordering are connected by a line. Performance of PCA was expected (Figure 2(a)), as the method projects examples onto a straight line. We can see that LLE failed to find a desired principal curve that would consist of two connected circles (Figure 2(b)). As can be seen in Figure 5(c), SC found better ordering than PCA and LLE and clearly separated upper and lower parts of the data set, but failed to find good ordering within the clusters. HC resulted in relatively smooth ordering with occasional jumps between circles, see Figure 2(d). We can see in Figure 2(e) that HC-olo reordered the HC tree to provide smoother ordering. As expected, LK had the shortest route, while the proposed TSP-means was very close to LK (Figures 2(f) and 2(g), respectively).

To compare the execution times for HC-olo, LK and TSP-means, we uniformly sampled 2-D and 3-D examples from a square and a cube of width 1, respectively, and increased data size from 500 to 1,000,000. In Figure 3 we show times in both logarithmic and linear scale in order to better illustrate the performance of algorithms for small- and large-scale data sets. In Figure 3(a) we can see that HC-olo method required prohibitively long processing time as n increased, and that it could not process data sets with more than few thousand examples. For 1 million examples TSP-means completed in around 20 minutes for both 2- and 3-D data, while for LK it took around 1 and 5 hours, respectively. We note that HC-olo and TSP-means scale linearly with the data dimensionality. On the other hand, LK algorithm uses k-d trees to speed up computations in lower-dimensions, but for higher-dimensional data the benefit of k-d trees decreases sharply. That is why results for LK in Figure 3 are representative only for 2- and 3-D data, while the execution time scaling for higher dimensions would be higher than shown and scale as $\mathcal{O}(n^{2.2})$. We did not run experiments for higher-D data since in that case the LK implementation we used requires distance matrix, which becomes infeasible for large n.

As observed in [48], although using LK on the whole data table results in the smallest tour length, it does not necessarily translate into the visually informative ordering. This can be explained by the fact that, unlike TSP-means, LK is not constrained by the underlying clustering structure in the data, rendering it sensitive to noisy examples. This is illustrated in Figure 4, where we generated 200 examples from each of the 2-D clusters representing 4 classes sampled from two-dimensional Gaussians centered at $(0, 0)$, $(0, 4.5)$, $(4.5, 0)$ and $(4.5, 4.5)$, with identity covariance matrices. We can see that LK achieved smaller tour length than TSP-means. However, TSP-means first visited examples from cluster 1, followed by cluster 2, then cluster 3, to finish by visiting examples from cluster 4 (see Figure 4(b)). On the other hand, in Figure 4(a) we see that LK jumped between clusters visiting them in the $\{3, 4, 1, 2, 1, 4, 3, 2, 3\}$ order, resulting in a lower quality of visualization. As discussed previously, this indicates that TSP-means accounts for clustering structure present in the data.

B. Validation of EM-ordering

In this section, we present performance of algorithms on benchmark labeled sets from UCI repository. Noisy *waveform* was created by appending 21 noise features to *waveform*. All data sets were normalized to zero-mean and unit variance. We limited ourselves to data sets of size $n = 1{,}500$ in order to be able to compare our method with resource-intensive SC and HC methods, and report average FOM after 5 experiments.

In Table III we report FOM performance of ordering methods on 11 classification data sets. Interestingly, in nearly all tasks the three lower-dimensional projection methods (PCA, LLE, and SC) were significantly outperformed by the competing techniques. EM-ordering was best on 7 out of 11 data sets, with the additional advantage of being much faster than

TABLE III. FOM SCORES FOR BENCHMARK DATA SETS (EM-1 AND EM-5 DENOTE EM-ORDERING AFTER 1 AND 5 ITERATIONS OF ALGORITHM 1, RESPECTIVELY, BASELINE RESULT IS FOM BEFORE REORDERING; ALSO SHOWN SIZE n, DIMENSIONALITY m, AND NUMBER OF CLASSES c)

data set	n	m	c	baseline	PCA	LLE	SC	HC	HC-olo	LK	EM-1	EM-5
iris	150	3	3	0.637	0.188	0.161	0.187	0.181	0.134	**0.114**	0.121	**0.114**
wine	178	13	3	0.649	0.225	0.531	0.056	0.062	0.056	0.045	0.047	**0.032**
breast	277	9	2	0.413	0.337	**0.324**	0.330	0.344	0.340	0.344	0.339	0.338
adult	1,500	123	2	0.374	0.267	n/a	0.235	0.276	0.267	0.252	0.238	**0.232**
banana	1,500	2	2	0.514	0.348	0.356	0.279	0.151	0.152	**0.147**	**0.147**	**0.148**
covertype	1,500	54	7	0.630	0.620	0.612	0.583	0.430	0.395	**0.382**	0.424	0.438
gauss	1,500	2	2	0.491	0.316	0.286	0.286	0.263	0.269	**0.260**	0.266	0.267
madelon	1,500	500	2	0.506	0.447	0.494	0.489	0.464	0.449	0.444	0.439	**0.399**
magic	1,500	10	2	0.464	0.416	0.451	0.360	0.258	**0.235**	0.243	0.244	0.259
waveform	1,500	21	3	0.680	0.462	0.461	0.461	0.266	0.250	0.249	0.239	**0.238**
wave noisy	1,500	42	3	0.680	0.472	0.493	0.466	0.344	0.309	0.310	0.282	**0.237**

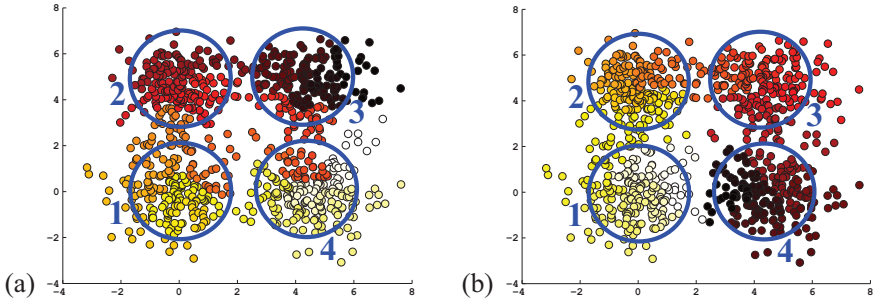

Fig. 4. FOM and L measures on a 2-D toy data set (color encodes the order of an example in the ordered data set, ranging from white to black): (a) LK (0.038; 181.44); (b) TSP-means (0.033; 199.86)

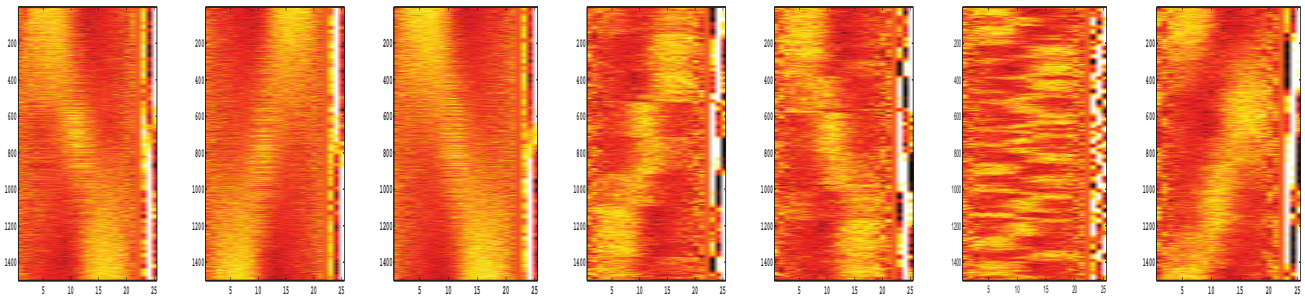

Fig. 5. Visualization of *waveform*, the last 3 columns are class assignments averaged over sliding window of length 20; FOM and L measures given in parentheses: (a) PCA (0.462; 7,231); (b) LLE (0.461; 7,211); (c) SC (0.461; 7,244); (d) HC (0.266; 5,265); (e) HC-olo (0.250; 4,817); (f) LK (0.249; 4,577); (g) EM-ordering (0.239; 4,921)

the closest competitors HC-olo and LK. Results after one iteration of Algorithm 1 were better than after five iterations in only two cases, indicating the benefits on feature scaling. Feature scaling was especially beneficial for *wine*, *madelon* and noisy *waveform*, where FOM dropped by the highest margin. Moreover, on *madelon* and noisy *waveform* data, for which irrelevant, noisy features were known beforehand, EM-ordering detected more than 90% of noisy features.

In Figure 5 we show heatmaps of reordered *waveform* data set, where rows correspond to examples, columns correspond to features, and color intensity represents a numeric value. As the data set has 3 classes, in the 3 rightmost columns we encode the class membership of examples. We averaged class membership over a column-wise sliding window of length 20 for easier interpretation of the results. Darker pixels indicate that examples within the window had the same label, thus indicating successful ordering. As seen in Figures 5(a), 5(b), and 5(c), lower-dimensional projection methods PCA, LLE, and SC obtained similar visualization results with very smooth-appearing heatmaps, but FOM results and the last three columns suggest that they did not provide compact grouping of examples with the same class labels. In contrast, HC and HC-olo resulted in better ordering, as shown in Figures 5(d) and 5(e), respectively. However, similarly to the results in Figure 2, tours often jumped between classes, reducing FOM and the quality of visualization. LK algorithm found the shortest path, as illustrated in Figure 5(f), which did not translate into good ordering. LK frequently jumped between classes, resulting in visually unappealing ordering. On the other hand, EM-ordering had the best FOM, as can be seen from Figure 5(g).

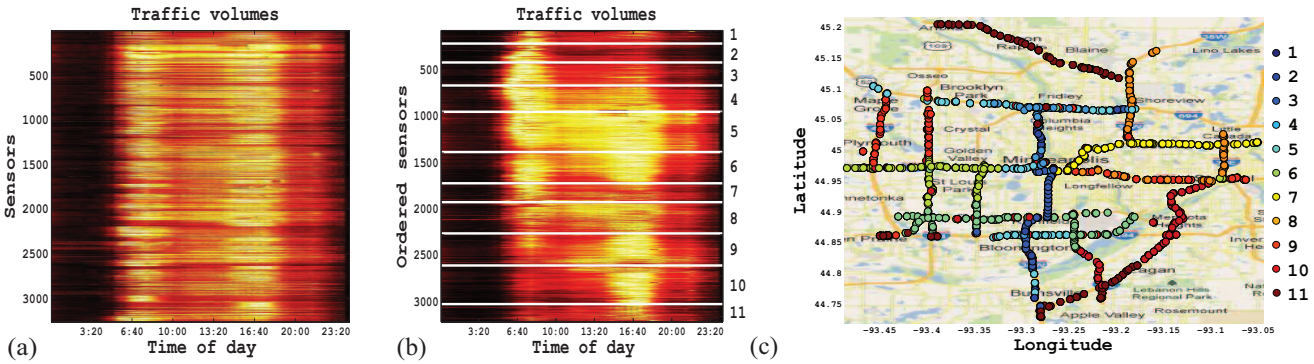

Fig. 6. Traffic data: (a) the original data set (brighter pixels denote higher volumes); (b) the ordered data set (white lines denote user-selected cluster boundaries); (c) color-coded sensor locations in Minneapolis road network (neighboring clusters were assigned similar colors)

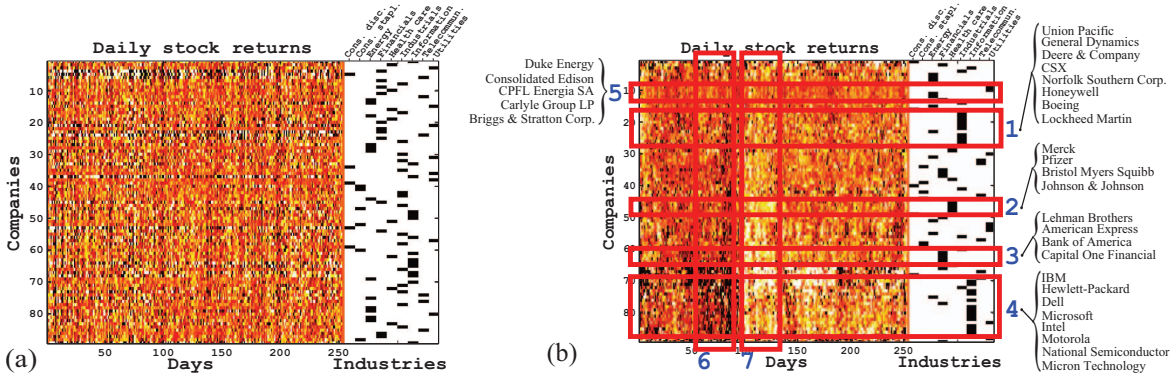

Fig. 7. *stocks* data set (9 rightmost columns encode industries; dark pixels in the heatmap encode high negative returns, while bright pixels encode high positive returns): (a) the original data set; (b) the data set after reordering and clustering upon inspection of the heatmap

C. Applications of EM-ordering

To illustrate the usefulness of ordering, we applied EM-ordering on two real-world data sets. The first is a set of traffic volumes (number of cars per minute) reported every 10 minutes by 3,265 sensors on highways in Minneapolis, MN, on December 23rd, 2005. The original, unordered data set is shown in Figure 6(a), from which little can be seen beyond presence of heavy traffic volume during early morning and late afternoon. In Figure 6(b) we show an ordered data matrix, where it becomes clear that there are several types of traffic patterns (heavy traffic during morning or afternoon only, light or heavy traffic during most of the day, etc.). To further illustrate the benefits of ordering, upon visual inspection we split the ordered sensors manually into 11 clusters. In Figure 6(c) we show the geographical locations of sensors, colored according to their cluster labels. We can see that the sensors along the same road segments were clustered together, and that nearby road segments were assigned to similar clusters. This example illustrates how ordering can be used for interactive exploration and visualization of data, which could be very useful to traffic engineers and planners. We note that this modestly-sized data set cannot be ordered using HC-olo and LK algorithms on a regular computer.

We also ran EM-ordering on *stocks* data set, representing 252 daily stock returns of 89 companies from 9 sectors of industry. Not much can be seen from the original data set shown in Figure 7(a) (the companies were sorted alphabetically). After reordering rows and columns of the data table interesting patterns emerged, as seen in Figure 7(b), which may provide useful insights to stock traders. We can observe that the ordering revealed several clusters of companies operating in industrials, health care, financials, and information technologies sectors (clusters 1 - 4, respectively), having specific patterns of daily returns. We also detected companies from energy and utilities sectors in cluster 5, whose daily returns, unlike returns of the companies from other sectors, did not fluctuate much. Lastly, after reordering the transposed data matrix (i.e., ordering days instead of companies), bear and bull trading days can be easily detected (clusters 6 and 7, respectively).

V. Conclusion

We proposed EM-ordering, an efficient reordering algorithm for data visualization, naturally emerging from entropy-minimization framework. In addition to finding a near-optimal ordering of the examples, the algorithm can automatically detect noisy features and decrease their influence on the final ordering. Moreover, unlike commonly used data visualization methods, our algorithm has very favorable time and space complexity, allowing efficient visualization of data tables with millions of examples. Empirical evaluation showed that the algorithm outperformed existing methods while being much faster, confirming that EM-ordering can present a powerful tool for analysis and visualization of large-scale data sets.

REFERENCES

[1] M. Friendly, "A Brief History of Data Visualization," in *Handbook of Computational Statistics: Data Visualization*, C. Chen, W. Härdle, and A. Unwin, Eds. Springer-Verlag, 2006, vol. III, pp. 15–56.

[2] D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler, "Challenges in visual data analysis," in *International Conference on Information Visualization*. IEEE, 2006, pp. 9–16.

[3] M. Friendly and E. Kwan, "Effect ordering for data displays," *Computational Statistics & Data Analysis*, vol. 43, no. 4, pp. 509–539, 2003.

[4] A. Inselberg, "The plane with parallel coordinates," *The Visual Computer*, vol. 1, no. 2, pp. 69–91, 1985.

[5] A. Inselberg and B. Dimsdale, "Parallel Coordinates," in *Human-Machine Interactive Systems*. Springer, 1991, pp. 199–233.

[6] M. Friendly, "Corrgrams: Exploratory displays for correlation matrices," *The American Statistician*, vol. 56, no. 4, pp. 316–324, 2002.

[7] S. S. Vempala, "Modeling high-dimensional data: Technical perspective," *Comm. of the ACM*, vol. 55, no. 2, pp. 112–112, 2012.

[8] S. T. Roweis and L. K. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[9] G. Hinton and S. Roweis, "Stochastic neighbor embedding," *Advances in Neural Information Processing Systems*, vol. 15, pp. 833–840, 2002.

[10] J. Bertin and M. Barbut, *Sémiologie graphique: les diagrammes, les réseaux, les cartes*. Mouton Paris, 1967.

[11] E. Mäkinen and H. Siirtola, "Reordering the reorderable matrix as an algorithmic problem," in *Theory and Application of Diagrams*. Springer, 2000, pp. 453–468.

[12] M. Hahsler, K. Hornik, and C. Buchta, "Getting Things in Order: An introduction to the R package seriation," 2007.

[13] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," *PNAS USA*, vol. 95, no. 25, pp. 14 863–14 868, 1998.

[14] Z. Bar-Joseph, E. D. Demaine, D. K. Gifford, A. M. Hamel, T. Jaakkola, and N. Srebro, "K-ary Clustering with Optimal Leaf Ordering for Gene Expression Data," ser. WABI '02, 2002, pp. 506–520.

[15] C. Ding and X. He, "Linearized cluster assignment via spectral ordering," in *International Conference on Machine learning*. ACM, 2004, pp. 30–37.

[16] S. Climer and W. Zhang, "Take a walk and cluster genes: a TSP-based approach to optimal rearrangement clustering," in *International Conference on Machine Learning*. ACM, 2004.

[17] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[18] D. A. Keim, "Visual exploration of large data sets," *Communications of the ACM*, vol. 44, no. 8, pp. 38–44, 2001.

[19] ——, "Information visualization and visual data mining," *IEEE Trans. on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 1–8, 2002.

[20] L. Wilkinson and M. Friendly, "The history of the cluster heat map," *The American Statistician*, vol. 63, no. 2, 2009.

[21] T. Loua, *Atlas statistique de la population de Paris*. J. Dejey & Cie, 1873.

[22] S. Vadapalli and K. Karlapalem, "Heidi matrix: Nearest neighbor driven high dimensional data visualization," in *ACM SIGKDD Workshop on Visual Analytics and Knowledge Discovery*. ACM, 2009, pp. 83–92.

[23] A. Tatu, F. Maass, I. Färber, E. Bertini, T. Schreck, T. Seidl, and D. A. Keim, "Subspace Search and Visualization to Make Sense of Alternative Clusterings in High-Dimensional Data," in *IEEE Symposium on Visual Analytics Science and Technology*. IEEE CS Press, 2012, pp. 63–72.

[24] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[25] G. J. Williams, P. Christen *et al.*, "ReDSOM: relative density visualization of temporal changes in cluster structures using self-organizing maps," in *International Conference on Data Mining*. IEEE, 2008, pp. 173–182.

[26] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[27] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, pp. 1373–1396, June 2003.

[28] Y.-H. Fua, M. O. Ward, and E. A. Rundensteiner, "Hierarchical parallel coordinates for exploration of large datasets," in *IEEE Conference on Visualization*. IEEE Computer Society Press, 1999, pp. 43–50.

[29] J. Walter, J. Ontrup, D. Wessling, and H. Ritter, "Interactive visualization and navigation in large data collections using the hyperbolic space," in *International Conference on Data Mining*. IEEE, 2003, pp. 355–362.

[30] A. O. Artero, M. C. F. de Oliveira, and H. Levkowitz, "Uncovering clusters in crowded parallel coordinates visualizations," in *IEEE Symposium on Information Visualization*. IEEE, 2004, pp. 81–88.

[31] W. M. F. Petrie, *Sequences in prehistoric remains*, ser. Reprint series in the social sciences. Bobbs-Merrill, 1899.

[32] J. Czekanowski, "Zur differential Diagnose der Neandertalgruppe," in *Korrespondenz-blatt der Deutsche Gesellschaft für Anthropologie, Ethnologie und Urgeschichte*, vol. XL(6/7), 1909, pp. 44–47.

[33] I. Liiv, "Seriation and matrix reordering methods: An historical overview," *Statistical Analysis and Data Mining*, vol. 3, no. 2, pp. 70–91, 2010.

[34] D. Guo, "Visual analytics of spatial interaction patterns for pandemic decision support," *International Journal of Geographical Information Science*, vol. 21, pp. 859–877, 2007.

[35] G. M. Del Corso and G. Manzini, "Finding exact solutions to the bandwidth minimization problem," *Computing*, vol. 62, pp. 189–203, July 1999.

[36] D. Blandford and G. Blelloch, "Index Compression through Document Reordering," ser. DCC '02, 2002.

[37] A. Pinar, T. Tao, and H. Ferhatosmanoglu, "Compressing Bitmap Indices by Data Reorganization," *ICDE05*, pp. 310–321, 2005.

[38] S. Roweis, "EM algorithms for PCA and SPCA," ser. NIPS '97. MIT Press, 1998, pp. 626–632.

[39] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[40] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Oper. Research*, vol. 2, pp. 393–410, 1954.

[41] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, 1958.

[42] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II, "An Analysis of Several Heuristics for the Traveling Salesman Problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977.

[43] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–563, 1976.

[44] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations*. Springer, 2002, vol. 12.

[45] D. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, and K. Helsgaun, "Certification of an optimal TSP tour through 85,900 cities," *Operations Research Letters*, vol. 37, pp. 11–15, 2009.

[46] K. Helsgaun, "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic," *European Journal of Operational Research*, vol. 126, pp. 106–130, 2000.

[47] W. T. McCormick, P. J. Schweitzer, and T. W. White, "Problem decomposition and data reorganization by a clustering technique," *Operations Research*, vol. 20, no. 5, pp. 993–1009, 1972.

[48] T. Biedl, B. Brejova, E. D. Demaine, A. M. Hamel, and T. Vinar, "Optimal Arrangement of Leaves in the Tree Representing Hierarchical Clustering of Gene Expression Data," Tech. Rep. CS-2001-14, 2001.

[49] P. A. DiMaggio, S. R. McAllister, C. A. Floudas, X.-J. Feng, J. D. Rabinowitz, and H. A. Rabitz, "Biclustering via optimal re-ordering of data matrices in systems biology: rigorous methods and comparative studies," *BMC Bioinformatics*, vol. 9, no. 1, p. 458, 2008.

[50] K. Y. Yeung, D. R. Haynor, and W. L. Ruzzo, "Validating clustering for gene expression data," *Bioinformatics*, vol. 17, no. 4, pp. 309–318, 2001.