Self-Sorting Map: An Efficient Algorithm for Presenting Multimedia Data in Structured Layouts

Grant Strong and Minglun Gong, *Member, IEEE*

***Abstract*—This paper presents the Self-Sorting Map (SSM), a novel algorithm for organizing and presenting multimedia data. Given a set of data items and a dissimilarity measure between each pair of them, the SSM places each item into a unique cell of a structured layout, where the most related items are placed together and the unrelated ones are spread apart. The algorithm integrates ideas from dimension reduction, sorting, and data clustering algorithms. Instead of solving the continuous optimization problem that other dimension reduction approaches do, the SSM transforms it into a discrete labeling problem. As a result, it can organize a set of data into a structured layout without overlap, providing a simple and intuitive presentation. The algorithm is designed for sorting all data items in parallel, making it possible to arrange millions of items in seconds. Experiments on different types of data demonstrate the SSM's versatility in a variety of applications, ranging from positioning city names by proximities to presenting images according to visual similarities, to visualizing semantic relatedness between Wikipedia articles.**

***Index Terms*—Algorithms, artificial neural networks, computational and artificial intelligence, computers and information processing, data visualization, neural networks, parallel algorithm, systems, man and cybernetics, user interfaces.**

I. Introduction

MULTIMEDIA data comes in many forms like text, images, and videos, to name a few. These types of data are normally high dimensional. Generally, high dimensional data can be hard for humans to understand in its raw form and trying to decipher the relatedness among nominal or non-numeric data can be even more challenging. Our visual and analytical systems are brilliantly engineered to decipher and extract patterns from complex pictures, yet they are ill-equipped to deal with lists of unorganized items. For this reason researchers have been trying for years to come up with better visualizations to facilitate data understanding and re-finding. Many powerful algorithms have been designed to reveal the underlying structures of the input data [26]. However, these techniques can require presenting data to users through complex interfaces, which limits their use to experienced users. Users may also have to explore the data through interactions, which sometimes are undesirable (e.g., display on mobile devices with small screens) or even unavailable (e.g., public display serving multiple users). On the other hand, when presenting a set of data to viewers, a simple yet widely-used approach is to place the data into a table. Such a 2D grid layout helps users to memorize the locations of particular data items and hence facilitates re-finding them later. However, without an intuitive way of sorting the data, finding a data item for the first time requires the users to linearly scan through the table, which can be time consuming.

Fig. 1 shows how weather in North America is presented using both an interactive map and a 2D grid. In the first case interaction (pan and zoom) is required due to occlusion whereas in the second understanding regional weather can be difficult. Our approach organizes cities based on their geographical locations. This facilitates users' memorization of the location of a given city in the grid aided by their prior geographical knowledge. Presumably this speeds up re-finding. In addition, since our result is free of occlusion, it makes full use of the screen space, allowing more information to be presented when the screen size is limited.

This paper presents a novel way of organizing and visualizing the data; see Fig. 1. First, assume a dissimilarity measure can be defined that tells us how related two data items are, thus giving an idea of how close they should appear. Our approach tries to generate a grid layout where the most related data items are placed together and the unrelated ones are pushed far apart. In addition, since the organization does not strictly rely on sorting the data or representing the data with vectors, it can be applied to a wide variety of forms of data, from high-dimensional vectors to nominal-valued items. Previous user studies have shown that such structured and similarity-based layouts can help users to find or re-find the desired images for image search tasks [10], [33].

More formally, this paper transforms the data organization and visualization problem into the following labeling problem: Let Ω be a set of data, for which a dissimilarity measure $\delta(\cdot,\cdot)$ is defined. Further assume that the dissimilarity satisfies the following constraints: $\delta(s,s) = 0$, $\delta(s,t) \geq 0$, and $\delta(s,t) = \delta(t,s)$ for $\forall s,t \in \Omega$. Now given a structured layout M with m ($m \geq |\Omega|$) cells, the objective is to assign each data item $s \in \Omega$ a unique cell L_s, such that the following normalized cross-correlation is maximized:

$$\underset{L}{\operatorname{argmax}} \sum_{\forall s,t \in \Omega} \frac{(\|P(L_s) - P(L_t)\| - \bar{P})(\delta(s,t) - \bar{\delta})}{\sigma_P \sigma_\delta} \quad (1)$$

Manuscript received February 11, 2013; revised July 25, 2013; accepted January 17, 2014. Date of publication February 13, 2014; date of current version May 13, 2014. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Vasileios Mezaris.

The authors are with Department of Computer Science, Memorial University, St. John's, NF Canada A1B3X5.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2014.2306183

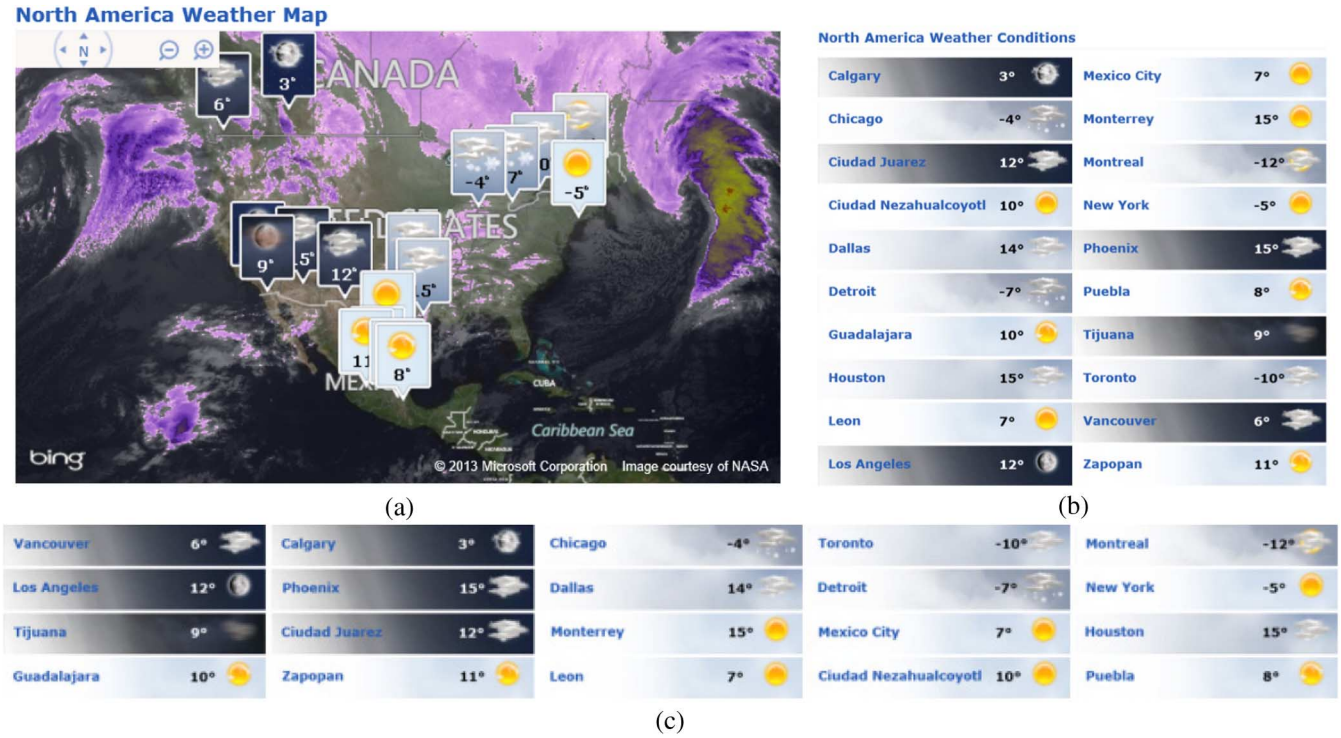

Fig. 1. Presenting the weather at 20 key cities in North America. Top row shows the two presentations used by the AccuWeather (http://www.accuweather.com/en/north-america-weather), whereas the bottom demonstrates the results of our approach. Directly positioning the cities using geographic coordinates (a) causes occlusions, forcing users to interact with the map. Sorting the cities lexicographically by their names into a 2D grid (b) allows quickly finding a particular city, but it is difficult to grasp the weather in a broader region. The proposed approach organizes the cities into a grid layout (c) while trying to preserve the proximity relationships, providing a better illustration of the regional weather trend than (b) without the need for interaction as (a) does. (a) Interactive map view. (b) Lexicographical list view. (c) 2D grid layout generated using Self Sorting Map.

where $P(x)$ is the location of cell x in the structured layout. $\bar{P}$ is the mean distance between any two assigned cells and $\bar{\delta}$ is the mean dissimilarity between any two data items. σ_P and σ_δ are the corresponding standard deviation values which normalize the correlation values to range $[-1, 1]$.

By definition, the above correlation measures how well the placements of data items, as a whole, correlate to the dissimilarities among them. Hence, maximizing the above equation yields a globally optimal solution where the distance between any pair of data items can best reflect their dissimilarities. That is, similar items are placed close together in the layout and dissimilar ones are placed far apart. Hence, the solution attempts to organize the data based on the topology defined by δ within the space and structural constraints of M. It is worth noting that even though the 2D rectangular grid layout is used throughout this paper, the above definition is valid for other lattice structures such as 3D grids and 2D hexagonal grids. The algorithm presented can be adapted to these structures as well.

If Ω is sufficiently small then it may suffice to evaluate every possible labeling assignment and determine the one giving the highest correlation. However, since the number of possible assignments is a function of the factorial of $|\Omega|$, the problem quickly becomes intractable as $|\Omega|$ increases. In these instances we need an efficient method that finds a near optimal solution. The algorithm discussed in this paper is such an approach. It searches for a locally optimal organization efficiently through a set of exchange operations.

II. Related Work

How to visualize a set of data based on their relatedness or similarity is widely studied. Some of these approaches try to present the data as graphs or networks [3], [6], [36], [39], where each node represents an item or a cluster of data and the edges represent connections between them. When the input data is highly interconnected, visualizing it using nodes and edges may yield busy graphs, making visual analysis of the relatedness among data a challenging task. In contrast, the presented work organizes data into a 2D grid and uses the distances among data locations to implicitly present the relatedness information. The resulting visual representation is similar to existing interfaces and is familiar to use (see Fig. 1 for example).

A. Dimension Reduction Techniques

Similarities in data can be visualized using dimensionality reduction techniques [2], [16], [20], [28], [38]. For example, being a set of manifold learning techniques, Multidimensional Scaling (MDS) [2], [20] is often used to map data onto 2D or 3D visual space. Given the function δ defining the dissimilarity between every pair of data items, the goal of MDS is to find a set of vectors $\{x_1, \ldots, x_{|I|}\} \in \mathbb{R}^N$ that minimizes the cost function:

$$\underset{\{x_1,\ldots,x_{|I|}\}}{\operatorname{argmin}} \sum_{\forall s,t \in \Omega} (\|x_s - x_t\| - \delta(s,t))^2 \qquad (2)$$

Intuitively, the set of vectors should model the relative similarity between the data items as positions in $\mathbb{R}^N$. Since this is posed as a least squares optimization problem, a solution can be found using variety of techniques; generally gradient descent is used. There are a host of variants to the MDS technique, a notable one is Sammon's mapping [28] that attempts to minimize a different non-linear error function:

$$\underset{\{x_1,\ldots,x_{|I|}\}}{\operatorname{argmin}} \frac{1}{\sum_{\forall s,t\in\Omega}\delta(s,t)} \sum_{\forall s,t\in\Omega} \frac{\left(\|x_s - x_t\| - \delta(s,t)\right)^2}{\delta(s,t)} \tag{3}$$

The key difference between the presented algorithm and the MDS based approaches is that ours tries to organize data into a structured layout. Instead of allowing a data item s to be placed at an arbitrary location x_s, we enforce it to be assigned to a unique cell L_s in the output structure. This transforms a continuous optimization problem into a discrete one. When used for visualizing the data, this feature enhances the ability of users to linearly scan the organized data because of its regular presentation.

When the input data can be represented as vectors in a high-dimensional space and the output is embedded in 2D or 3D space for visualization, the MDS problem is considered to be a Multidimensional Projection (MDP) problem [22]. A well-known MDP technique is the Self-Organizing Map (SOM), which indirectly maps input vectors of N dimensions to an output space that is the same dimension as the structure of the map [17], [18]. An SOM consists of a network of interconnected units, each holds an N-dimensional, randomly initialized, weight vector. The weight vectors at different units are trained using competitive learning. When a randomly selected input vector is fed to the SOM, its Euclidean distance to all weight vectors is computed and the best matching unit (BMU) is found. The weights of the BMU and its neighbors are then adjusted towards the input vector. Once the training is complete, all input vectors are mapped to the location of their BMUs, providing an intuitive visualization for multi-dimensional data [5], [11], [24].

B. Occlusion Removal Approaches

There is a family of techniques for creating occlusion free layouts from the organizational results like those from MDS [7], [8], [29]. These techniques treat the items being visualized as objects with size rather than dimensionless points. They post-process the layout in such a way that error is minimized under the constraint that nothing overlaps. There is no stipulation that the items conform to a regularized layout. Their positions can be augmented freely as long as error is low.

Dwyer *et al.* propose an algorithm that performs a constraint optimization across the graph of placed items [7]. It first generates a system of constraints (inequalities based on the positions) between nearest neighbors along scan lines of the x-axis. It determines a solution for the constraints and runs the same procedure on the y-axis. The result is a version of the input graph without occlusion.

A problem with the above method is its tendency to stack items because of the independent processing of the two axes. This was addressed by using Rolled-out Wordles [29], which aims to produce similarly occlusion free results without breaking the orthogonal ordering. It does this by sorting the items linearly or concentrically and then placing the items in that order. If an overlap is encountered, the overlapping item is spiraled from its original place until a clear position is found. The result is a tight ordering that closely models the original topology.

A final relevant method of overlap removal we will discuss here has the ability to generate layouts akin to the structured layouts that our Self-Sorting Map algorithm works on. It is called MIOLA [8] and it uses mixed integer optimization on non-overlap constraints similar to those previously mentioned [7]. The layout can have varying degrees of structure, the most rigid being a grid.

In summary, these are post-processing techniques that work on a set of 2D locations generated by other dimension reduction methods. In contrast, our approach starts from the source and takes the dissimilarity matrix along the data items as input.

C. Occlusion Free Data Organization Approaches

Approaches have also been proposed to achieve similar objectives to ours, namely organizing data in an occlusion free and structured way. The Hexboard [5], [24] is an example of such an approach. It seeks to connect items based on similarity as it incrementally grows a hexagonal grid using a greedy heuristic. Each new item is placed by finding a similar item that exists in the grid and then finding a position for the new item in the vicinity of this similar item. The new position needs to minimize error with respect to other positioned items. Older items might be shuffled from their positions if the new item is a better fit. The shuffled item then follows the same procedure to find a new position for itself. Eventually whichever item that is in movement will find a free spot in the grid. Item movements are made if the score of a metric computed from neighboring items is better. The neighborhood can include all items that have been placed on the board, dubbed "full mode", or it can include a randomized subset of neighboring items, dubbed "fast mode". Full mode produces the best results but is only feasible for smaller datasets. The hexagonal structure that items are being placed in is not constrained, allowing it to grow into arbitrary shapes. The merit of such a design is handling dynamic datasets though incremental updates. It differs from our approach which focuses on efficiently arranging static datasets in a constrained layout.

The approach we present is entitled the Self-Sorting Map (SSM) due to the commonalities it bares to the Self-Organizing Map algorithm. Given a set of data, both approaches map them onto a 2D or 3D structure while trying to preserve the topology of the input data as much as possible. Hence, both can be used to visualize a high-dimensional data space. However, the SSM avoids the explicit use of the vectors, making it possible to handle nominal data. In addition, it guarantees to assign each data item to a unique unit in the map, whereas SOM often returns the same BMUs for multiple data items which requires post-processing, like the use of a k-d tree [34], to avoid occlusion when the results are visualized. If a minimalistic grid free of occlusion is desired, the SSM can produce results that are of equivalent or better quality, in terms of correlation and visual inspection, than the SOM at a far lower computational cost.

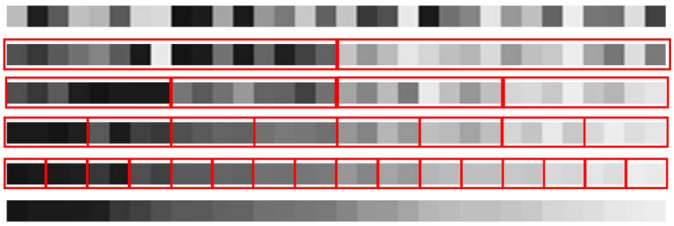

Fig. 2. Self-sorting on 1D array whose values are represented using intensities: Given the initial random array (top), the algorithm goes through multiple stages with decreasing block sizes. After each stage, the cells at the same position of different blocks are sorted. The full array is sorted once the block size reaches one (bottom).

Finally, compared with our preliminary work on this topic [31], the SSM algorithm discussed here is further enhanced for better organization results (Section III.B), lower computation cost (Section III.D), and more user controls (Section III.E). We also implement the algorithm in parallel on a Graphics Processing Unit (GPU), with implementation details and performance gains presented in Section 4. Lastly, more experimental results demonstrating different applications of the SSM algorithm are provided in Section 5.

III. Self-Sorting Map

To facilitate the understanding of the presented algorithm, here we first explain how it works for the simple case where the objective is to position a set of numbers into a 1D array. We then discuss how to use the algorithm to organize a general dataset into high-dimensional structures.

A. Sorting Numbers into a 1D Array

When the objective is to organize a set of numbers into a 1D array so that similar numbers are positioned together, we can obtain the optimal solution by simply sorting all the numbers, in either ascending or descending order. Being a fundamental computer science and mathematics problem, sorting has been extensively studied. There are many classic and efficient algorithms, such as quicksort and mergesort, but none of them can be easily extended to organize an arbitrary set of data into high-dimensional structures. The SSM, on the other hand, is designed for handling general cases, even though it is not as efficient for sorting numbers into a 1D array.

As shown in Fig. 2, given a set of numbers initially randomly placed inside a 1D array, the SSM first splits all cells into 2 blocks. Numbers in the first block are paired up with the ones at the corresponding cells of the second block. The two numbers in each pair (s, t) are compared against each other and an exchange is performed if $s > t$. After all pairs are processed in parallel, the first stage is completed.

The second stage further splits each block in two, resulting in four smaller blocks. Corresponding cells in adjacent (even, odd)-numbered blocks are first compared and swapped if necessary, followed by corresponding cells in adjacent (odd, even)-numbered blocks. For example, block 1 and 2 are paired with each other first for swapping and then block 2 and 3 are paired. The even-odd swap and odd-even swap alternates until the process converges, i.e., all data at the corresponding cells of different blocks are sorted. The process then continues to the next stage by dividing each block into two, until the final stage is reached where all blocks contain one cell only.

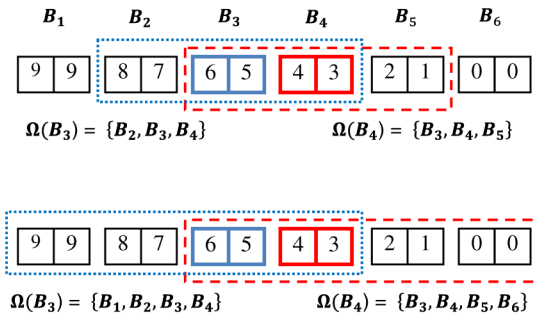

Fig. 3. The neighborhoods used for target calculation when swapping data between two paired blocks B_3 and B_4. Our previous approach uses 3-block neighborhood (top). In our revised approach, each block's neighborhood contains 4 blocks, which is offset away from the block it is paired with (bottom). The values of the targets T_3 and T_4 are given below.

The above process guarantees the final result is a sorted array. If we only look at the final stage, where each block contains just one cell, performing even-odd and odd-even swap until convergence is essentially the odd-even sorting algorithm. However, our approach is more efficient than odd-even sorting since it incorporates the basic idea of shell sort, i.e., using an increment sequence to allow the comparison and exchange of elements far apart. The increment sequence used here, $\{1, 2, 4, 8, \ldots, 2^k\}$, fits well for parallel implementation, even though it is known to be less efficient than other candidates, such as $\{1, 4, 10, 23, 57, \ldots\}$ [4].

B. Organizing High-Dimensional Data

High-dimensional data, such as samples from the RGB color space, cannot be meaningfully sorted—although one could sort colors lexicographically based on the red channel first, then the green and blue channels, such a sorting cannot guarantee similar colors will be placed together. To organize high-dimensional data, we make the following changes to the above baseline algorithm.

Since there is no ordering defined among high-dimensional vectors, given a pair of data (s, t), whether exchange is needed cannot be based on the comparison $s > t$. Instead, the decision here is based on whether an exchange can reduce the total difference between the two data items and their neighbors. To perform this evaluation efficiently, we first compute a target vectors T_i for each block of cells B_i using:

$$T_i = \frac{1}{|\Omega(B_i)|} \sum_{\mathrm{B_j} \in \Omega(B_i)} \frac{\sum_{s \in B_j} s}{|B_j|} \tag{4}$$

where $\Omega(B_i)$ is the neighborhood defined for block B_i and s is any item from the cells of B_j; see Fig. 3 for an illustration. In essence, this equation computes the target T_i of block B_i as the average of the data items stored at the blocks inside B_i's neighborhood.

As shown in Fig. 3, our preliminary approach uses Gaussian-weighted average of a centered 3-block neighborhood [31]. Here we let each neighborhood include an

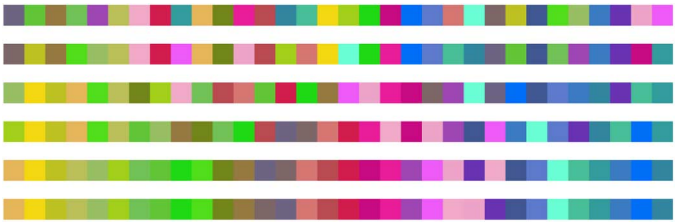

Fig. 4. Organizing data without using ordering information. The top row shows the initial (random) arrangement. The remaining 5 rows show the intermediate results after each of the 5 stages.

extra block away from the paring block, which leads to diversification among the computed targets and allows faster convergence. The Gaussian-weighted average is also replaced with simple average to allow efficient calculation on GPUs. It is also worth noting that, to allow the targets of different blocks to be calculated using different sets of neighbors, we start with splitting the cells into four blocks instead of two.

Once the targets are calculated, the decision on whether a swap is needed for a given pair (s, t) from adjacent blocks, $s \in B_i$ and $t \in B_{i+1}$, is based on if the following expression can be minimized after swapping:

$$\underset{(s,t)}{\operatorname{argmin}}(\|s - T_i\| + \|t - T_{i+1}\|) \tag{5}$$

where $\|a - b\|$ computes the Euclidean distance between two vectors a and b.

The above test (and swap if necessary) can be performed for all pairs in parallel. The target vectors are then updated based on the new data layout. The processes of computing targets and swapping are alternated until a convergence is reached, i.e., no more swaps are available and all target vectors stay constant.

Careful readers may notice the similarity between the above iterative process and the k-means clustering algorithm [15]. Both approaches alternate between finding the mean of each cluster and rearranging data into appropriate clusters. As a result, this self-organizing mechanism congregates input data with the means which they are similar to. This in turn strengthens those means, leading to a greedy convergence. A difference in k-means is that it does not impose any constraints on data moving into (or out of) clusters, meaning different clusters may have different numbers of data items. Our approach on the other hand, only allows swapping between blocks, which ensures that all cells are occupied by at most one data item.

Fig. 4 shows the results of organizing both grayscale and color vectors into a 1D array using the approach discussed above. It demonstrates how data is rearranged without requiring strict ordering information. Note that in Fig. 4(a) grayscale vectors are sorted by intensity even though no explicit intensity comparison is performed.

C. Organizing Nominal-Valued Data

Some datasets, such as a collection of Wikipedia articles, are not real-valued, and hence, the targets cannot be computed based on the above mean-based calculation. Assuming the dissimilarity $\delta(s, t)$ is defined for any given two data items s and t, here we discuss how to handle nominally valued data based only on $\delta(\cdot, \cdot)$.

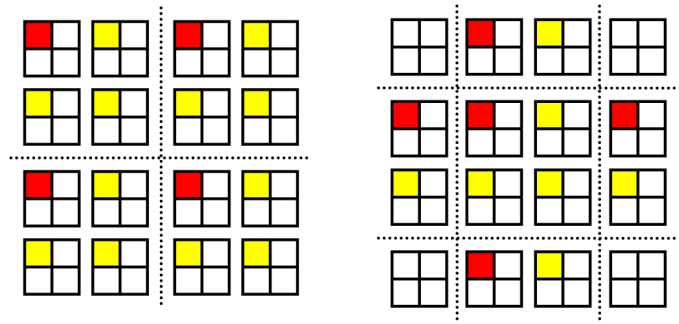

Fig. 5. Splitting a 2D map into 4×4 blocks and grouping blocks using even-odd (a) and odd-even (b) settings. In both subfigures, the red cell is matched to the three yellow cells from the grouped blocks.

The basic idea is to find a data item that best represents a given block B_i and use it as the target T_i. More formally, we compute the target as the data item inside the neighborhood of B_i that has the minimum total dissimilarity to other data items in the neighborhood $\Omega(B_i)$:

$$T_i = \underset{t \in \Omega(B_i)}{\operatorname{argmin}} \left(\sum_{s \in \Omega(B_i)} \delta(s, t) \right) \tag{6}$$

Note that the target can be any data item within the neighborhood $\Omega(B_i)$. In our preliminary work [31] the target item could only be chosen from among the data items in B_i alone. Expanding the target search has two major benefits. Firstly, more suitable targets can potentially be found. Secondly, we can avoid the special treatment of performing neighboorhood target selection at the final block level alone (to enable movement) like the original approach [31] required. Allowing the target to be selected from anywhere in the neighborhood like we do here renders the final stage of the algorithm just like any other.

D. Handling 2D Structural Layouts

The algorithm posed so far can organize an arbitrary dataset into a 1D array as long as a dissimilarity measure is defined. Now let us extend the algorithm further for organizing data into a 2D grid. The same principles can also be used for organizing data into other lattice structures like a 3D grid or a 2D hexagonal grid.

When handling a 2D grid, we start by randomly filling cells with items and then splitting all cells into 4×4 blocks, each of which will be further split into four smaller blocks in the next stage. We then group even-indexed blocks with odd-indexed blocks along both X and Y directions; see Fig. 5(a). This is followed by a shifted grouping in which odd-indexed blocks are grouped with even-indexed blocks along both directions (Fig. 5(b)). Alternating between these two block group settings allows a given block to swap data with its four nearest neighbors, facilitating data moving toward the desired cells. Please note that in [31], the block groupings are shifted independently on the X and Y axes, resulting three different block grouping configurations. Here we combine the two shifted groupings into a single one by shifting in both the X and Y axes at the same time to avoid biasing an axis. As a result, only two block grouping

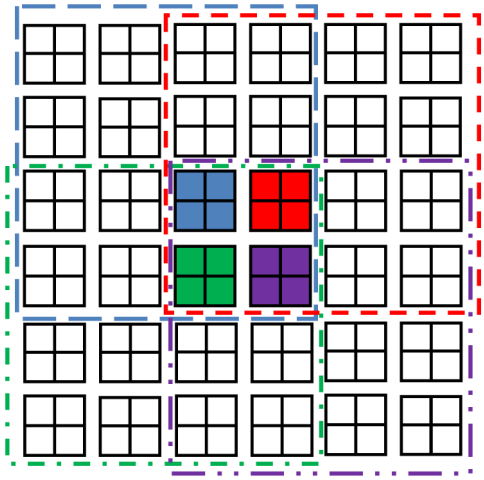

Fig. 6. The neighborhoods defined for the four color-coded paired blocks in the center. Each neighborhood encompasses a 4×4 block window that is offset away from the other blocks in the same group.

```
Randomly place data into the layout as an initial
arrangement;
while block size > 1 do {
   Split each block into 4 smaller blocks;
   do {
      for each of the two block groupings
         (even-odd and odd-even) {
         for each block B_{i,j} (i,j ≤ n) do
            Update the target T_{i,j} for B_{i,j};
         for each set of 4 blocks do {
            Group each cell s in the 1st block with
cells in the remaining 3 blocks;
            for every quadruple (s,t,u,v) do {
               Find the arrangement that minimizes
the distance between the items and the targets;
               Perform exchange if needed;
            }
         }
      }
   } while exchanges are still available and the
maximum number of iterations (L) is not reached;
}
```

Fig. 7. Pseudocode for the Self-Sorting Map algorithm.

configurations are needed, leading to less computation per iteration while achieving equivalent results.

Once the blocks are grouped, we determine the neighborhood $\Omega(B_i)$ for each block B_i, which is used for computing the target T_i. Similar to the 1D cases, here we use windows that are centered away from the grouped blocks; see Fig. 6. With the neighborhood determined, the target T_i for each block B_i is computed using either Equation (4) or (6) depending on the type of data to be organized.

The next step is to swap data between grouped blocks using the targets as guides. Here data items in the corresponding cells of the grouped blocks form quadruples. For example, the data item in the red cell shown in Fig. 5(a) is grouped with the ones in the three yellow cells to form a quadruple. The organizations for all quadruples are handled independently in parallel. To place a given quadruple (s, t, u, v) into four cells, there are $4! = 24$ possible alignments. To avoid being tripped into local minimum, here we simply enumerate all 24 possibilities and find the one that minimizes the following:

$$\underset{(s,t,u,v)}{\operatorname{argmin}} \begin{pmatrix} \delta(s, T_{i,j}) + \delta(t, T_{i+1,j}) + \\ \delta(u, T_{i,j+1}) + \delta(v, T_{i+1,j+1}) \end{pmatrix} \tag{7}$$

Fig. 7 summarizes the final SSM algorithm in the form of pseudocode. To avoid verbose description, here the size of output grid is assumed to be $N \times N$, where N is power of two. The extension for non-power-of-two grid requires special consideration for leftover blocks and blocks with fewer cells, whereas extension for non-square grid may involve an additional alignment stage that swaps along only X or only Y axis. Organizing datasets that do not have enough items to fully fill the map is also possible. In these cases, the algorithm fills the empty cells with placeholders and does not consider them in the alignment cost, i.e. δ in Equation (7) will be zero if either argument is a placeholder.

Fig. 8 shows the state of a 64×64 map of 4096 Lab color vectors after the algorithm has finished working at each block size. For quantitative evaluation, the cross-correlation values, as defined in Equation (1), are calculated and shown in the

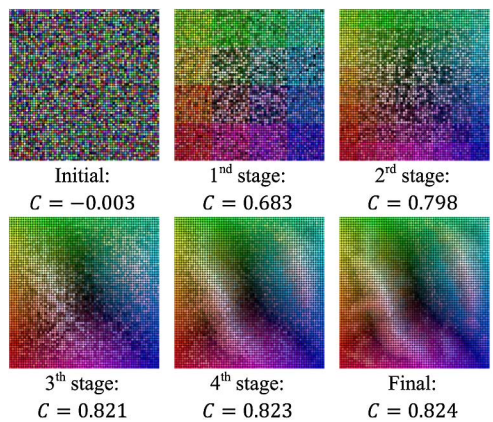

Fig. 8. The organization of 4096 Lab color vectors after different stages. The corresponding correlation scores are given below the images.

figure. As expected, the correlation is close to zero for the initial layout where the color vectors are randomly placed. The correlation steadily increases as the organization goes through different stages, suggesting that the exchanges performed improve the organization. The final score reaches a high positive correlation between the colors and their positions in the structured layout. To show the effect of randomness on organization quality, Fig. 9 presents the histogram of correlation scores from 100 runs of the SSM on this dataset with different initializations and swap orders. The histogram shows that with different random numbers, the correlation scores may vary about 2%. For comparison, the histograms for the SOM organization and for the original SSM [31] are also shown, which suggest that the revised SSM approach can produce statistically better organization than both the SOM and the original SSM.

E. Enforcing Boundary Conditions

The algorithm discussed up until now does not provide users with any means to control the organization. In some applications, users may desire to have data arranged in particular ways. For example, when positioning different cities

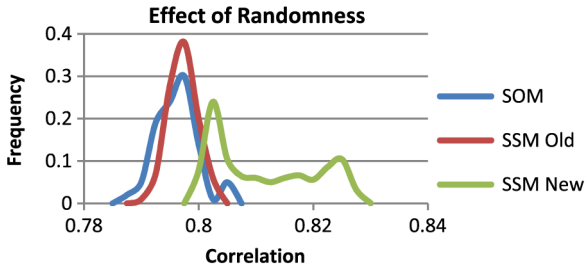

Fig. 9. 100 organizations of a 64×64 map of colors by a SOM, the original SSM [31], and the updated SSM proposed here. The initialization and item swap orders are random. The distributions are 0.7948 ± 0.0035, 0.7959 ± 0.0025, and 0.8101 ± 0.0089, respectively.

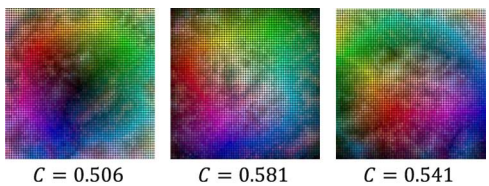

Fig. 10. Organizations generated using additional constraints on the border of the map. Left: using white color as boundary condition attracts bright colors to the boundaries and pushes dark colors to the center. Middle: using black as boundary condition has the opposite effect. Right: using white on top and black on bottom attracts colors with different brightness to different sides. Note that the correlation scores are much lower than the one obtained without any boundary constraints.

in a 2D grid (see Fig. 1), it might be more intuitive to have cities on the northern hemisphere placed on the top of the grid, even though flipping the grid upside down does not affect the cross-correlation score. Now we will show how users can influence the organizational results by simply adding boundary conditions under the proposed SSM approach. In constrast, when using dimension reduction techniques such as SOM and MDS, controlling the results can generally be difficult, unless more recent techniques specifically designed for user interactions are used [14], [23].

To achieve the desired organizational results, we allow users to specify data items outside the 2D grid. These items do not participate in swapping, but they are used during the target search calculation and hence influence the targets generated for blocks near the border. Consequently, input data items that are similar to the specified data items will be attracted toward the corresponding border of the 2D grid. As shown in Fig. 10, by enforcing different boundary conditions, we can organize the same set of color samples differently.

IV. Parallel Implementation

As is evident from Section 3, the Self-Sorting Map algorithm can be broken down into two logical stages; computing a target for each block and then swapping data between blocks based on those targets. Both stages can be performed using parallel kernels on streaming multiprocessors (SMPs), like those available in GPUs. In this section we will discuss how to design the parallel kernels, as well as analyze the time complexity of the algorithm.

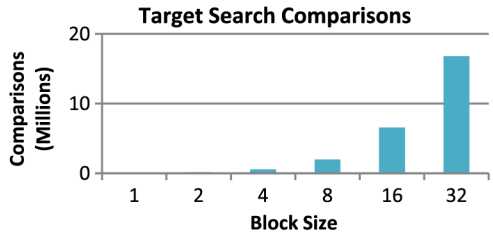

Fig. 11. The number of comparisons needed for computing the true centroid targets of different block sizes increases dramatically as the number of data items in each block does.

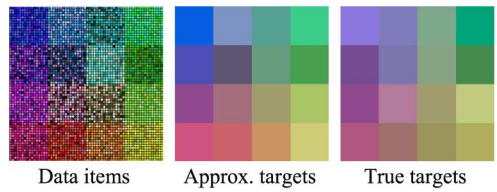

Fig. 12. The first image is the original data after items have been swapped into 16×16 blocks. The middle one shows the approximate centroids of those blocks' neighborhoods. The last image shows the corresponding true centroids calculated directly using Equation (6). The approximate centroids are similar yet more vibrant.

A. Target Generation Kernel

Taking the current data layout as input, the target generation kernel computes a target for each block of data items. The actual calculation involved depends on whether the mean or centroid is used. Here we will discuss how to compute mean targets first, followed by the centroid targets.

Since the neighborhoods defined for adjacent blocks overlap each other, directly calculating the mean target using Equation (4) results in redundant computations. A more efficient approach is to pre-compute the means of all blocks using only the data items inside the blocks. The result forms a 2D mean map whose resolution depends on the current block size. The mean map is then used to calculate the average in each block's neighborhood. To utilize the parallel processing power of SMPs, here the mean map is generated using parallel reduction. That is, we first calculate a mean map for blocks of 2×2 cells, which are then used to calculate a mean map for 4×4 sized blocks until the current block size is reached.

Compared to mean target calculation, centroid target searching is a much more expensive operation. For example, if we are given a map of 64×64 data items and want to find the target for a 16×16 sized block, we have to search all of the items in that block and the neighboring blocks to find the item that has the minimum total dissimilarity among all of the others, leading to millions of comparisons (see Fig. 11). Borrowing ideas from the mean target calculation, we can mitigate a lot of this complexity by computing approximate targets rather than the true ones. That is, since far fewer operations are required to compute the centroids of smaller blocks than of larger ones, we find the approximate centroid of a given block by repeatedly performing a centroid search on 2×2 sized blocks in a parallel reduction fashion; see Fig. 12. The approximate centroids for all blocks form a centroid map, which is then used to find the

approximate target of each block, i.e., the centroid of centroids of the neighboring blocks.

B. Data Swapping Kernel

Once the targets are determined, the second stage is carried out by a single kernel that swaps items between grouped blocks in such a way that dissimilarity between the items and the targets of those blocks is minimized. To do this, the kernel looks at each item in parallel in the context of a quadruple of four items formed by taking one item from each of four grouped blocks. Items chosen for each quadruple only ever belong to one such quadruple so as to maintain mutually exclusivity and thus the ability to have items swapped between blocks in parallel without conflict. Once four data items are grouped, they are swapped using the procedure described in Section 3.3, i.e., the four items are aligned with the four targets by checking all 24 possible combinations to find the one yielding the lowest total dissimilarity.

The two stages of finding targets and swapping items occur in this order and are repeated together as the algorithm dictates. Eventually the targets come to represent the items in their blocks well and few swaps need to be done. At this annealing point the algorithm moves on to the next block size where the map is considered to have more blocks that contain fewer items. The kernels continue to work as defined even with changing input sizes.

C. Complexity Analysis

To organize a set of N data into a $\sqrt{N} \times \sqrt{N}$ map, the presented SSM algorithm needs to go through $\log_2 \sqrt{N}$ stages. On the k^{th} stage, the data is split into 4^k blocks with each block containing $N/4^k$ cells. Finding the mean for each block requires $O(N/4^k)$ operations, whereas finding the true centroid requires $O((N/4^k)^2)$. Hence at the k^{th} stage, the total number of operations needed for target generation of all 4^k blocks is $O(N)$ for mean and $O(N^2/4^k)$ for centroid. Regardless of whether mean or centroid targets are used, the data swapping step requires $O(N)$ operations since each data item is processed only once.

The target generation and data swapping steps are repeated until convergence or the maximum number of iterations, L, is reached. The number of iterations required may vary per application since different datasets may converge at different times. Hence, we simplify the analysis here and consider the worst case time complexity of the SSM algorithm, where L iterations are used at all stages. Under this scenario, the serial SSM algorithm using mean targets requires $O(L \cdot N \cdot \log N)$ operations to complete all $\log_2 \sqrt{N}$ stages. When the centroid is used, the time complexity can be computed as:

$$LN^2 + \frac{LN^2}{4} + \frac{LN^2}{4^2} + \cdots + LN = O(L \cdot N^2) \quad (8)$$

Now assume that we have a parallel computer with N processors. On the k^{th} stage, where each block contains $N/4^k$ cells, the time needed for computing the mean targets or the approximate centroid targets using the aforementioned parallel reduction approach is $O(\log(N/4^k))$, whereas the time required for data swapping is $O(1)$. Hence, the time complexity for the fully parallel version of the algorithm is:

$$L\log(N) + L\log\left(\frac{N}{4}\right) + L\log\left(\frac{N}{4^2}\right) + \ldots + L = O(L \cdot (\log N)^2) \quad (9)$$

The speedup of the parallelization is $O(N/\log N)$, with the efficiency being $O(1/\log N)$. Section V.C shows how these theoretical runtimes play out on real hardware.

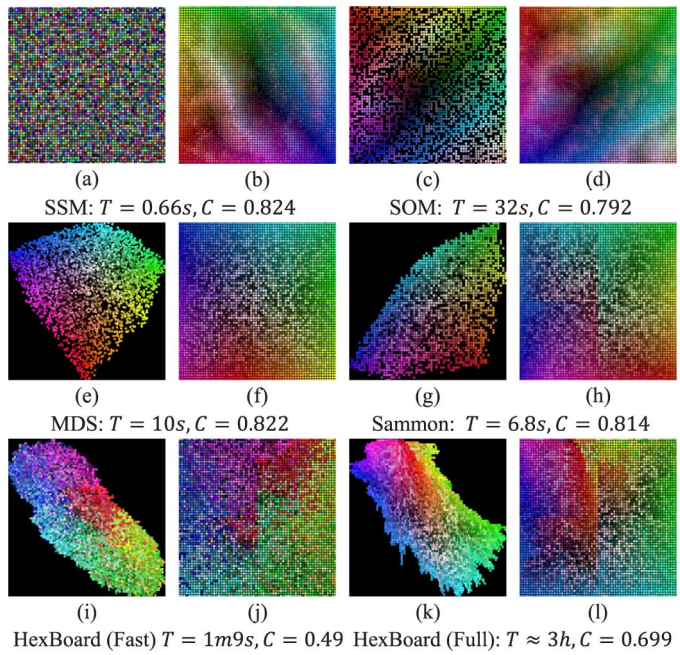

Fig. 13. Organizations of 4096 Lab color vectors. (a) is the starting set of items in all cases and (b) is the result from the SSM. For comparison, results of some related techniques are shown after. For these other methods, the original result is given first (images (c), (e), (g), (i), (k)), followed by the corresponding aligned versions (images (d), (f), (h), (j), (l)) generated using k-d trees. The total organization time and final correlation between position and dissimilarity across the grid aligned maps are given below the respective images.

V. Results

In order to exercise the proposed algorithm, we test the SSM against related techniques and varying data types, including Wikipedia articles, images, and cities. We investigate how the organization can be influenced by attaching a priori information to the border of the map. We also look at the performance figures of the SSM algorithm when it is executed on both a CPU and a GPU.

A. Comparison with Existing Approaches

For comparison to the SSM, we run existing methods on the same data. Fig. 13 shows the results of organizing a set of 4096 Lab color vectors into 64×64 cells in each of these cases. The Lab color space is used since the Euclidean distance between Lab color vectors represents the perceived similarity between those colors. The result of the SSM is obtained using mean targets with the maximum number of iterations per stage set to $L = 4$. The Sammon's mapping is obtained using the Sammon projection component of the HiSee [12] and the result of the SOM is generated using our own implementation [11], [24]. The

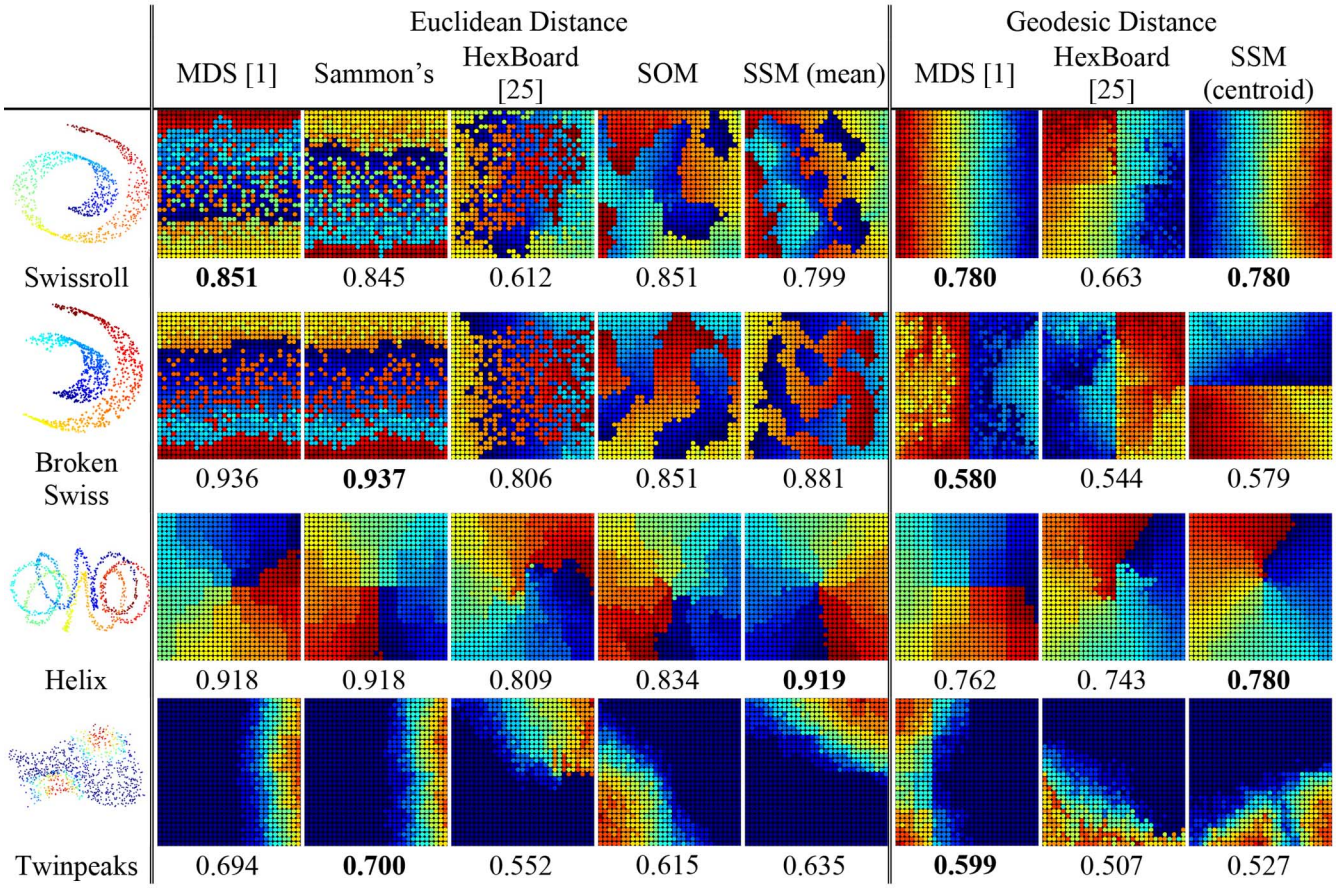

Fig. 14. Results of different approaches on four artificial datasets, each of which contains 1024 color-coded 3D vectors. The original datasets are shown in the left column, followed by organizations generated using existing dimension reduction methods with k-d tree alignment and the SSM under two different distance measures. The corresponding correlation score is given below each figure. The highest correlation scores are shown in **bold**. Note that results of Sammon's mapping under geodesic distances are not shown since they resemble MDS in appearance and correlation.

results of the HexBoard under both "fast mode" and "full mode" are shown, which are generated using the authors' code [5], [24].

Visual inspection confirms that data overlapping occurs when using dimension reduction techniques; see Fig. 13(c), (e), and (g). To uncover the occluded data, we need to either use a zoom and pan interface or apply an additional occlusion removal process [7], [8], [29]. Since there are a power-of-two data items and a complete binary tree can be built, we can align the results into a grid by using a k-d tree [34]. Nevertheless, artifacts (random and isolated dots) and noticeable seams show up in the result of Sammon's mapping (Fig. 13(h)), which is understandably not conducive to a grid layout since it arranges things in continuous space. The result of the SOM after alignment (Fig. 13(d)) is visually similar to the one for the SSM, which is likely due to the fact that both techniques incorporate a type of annealing in their organizational neighborhoods over time. However, careful inspection shows that the SSM result pushes saturated colors to the edges of the map whereas the SOM places grey color near the bottom right corner. This makes the result of SOM less ideal at coarse level. In addition, the SOM requires more computation for unsupervised training and an additional k-d tree post-processing to eliminate occlusion, whereas the SSM directly generates the desired result in the shortest time among the three approaches.

While the results of the HexBoard approach are occlusion free, they are not constrained to the grid layout. Among its two computation modes, the result of the "fast mode" (Fig. 13(i)) is quite a bit noisier than the "full mode" (Fig. 13(k)), which requires hours to compute. Applying k-d tree based alignment can transform HexBoard results into grid layout (Fig. 13(j) & (l)), but yields noticeable seams as in the case for the Sammon's mapping. It is worth noting that the HexBoard is designed for incrementally handling dynamic datasets, a merit that is not shown in this test.

The correlation score C is also used here to quantitatively measure the correlation between similarity and proximity. The measurements agree with the visual evaluation. That is, the SOM has a low score due to the less than ideal color layout at coarse level. The results of MDS and Sammon's mapping are noisier at fine level than the one from the SOM, but has better color layout at coarse level, i.e., the saturated colors are pushed to the edges. Hence it has a higher C score. The result of SSM also pushes saturated colors to the edges and is smoother at fine level than Sammon's mapping here. This leads to the highest C score.

Next, we evaluate how these algorithms deal with different topological structures in the data using four standard artificial datasets like those in [37]. The "swiss roll" is a 2D manifold

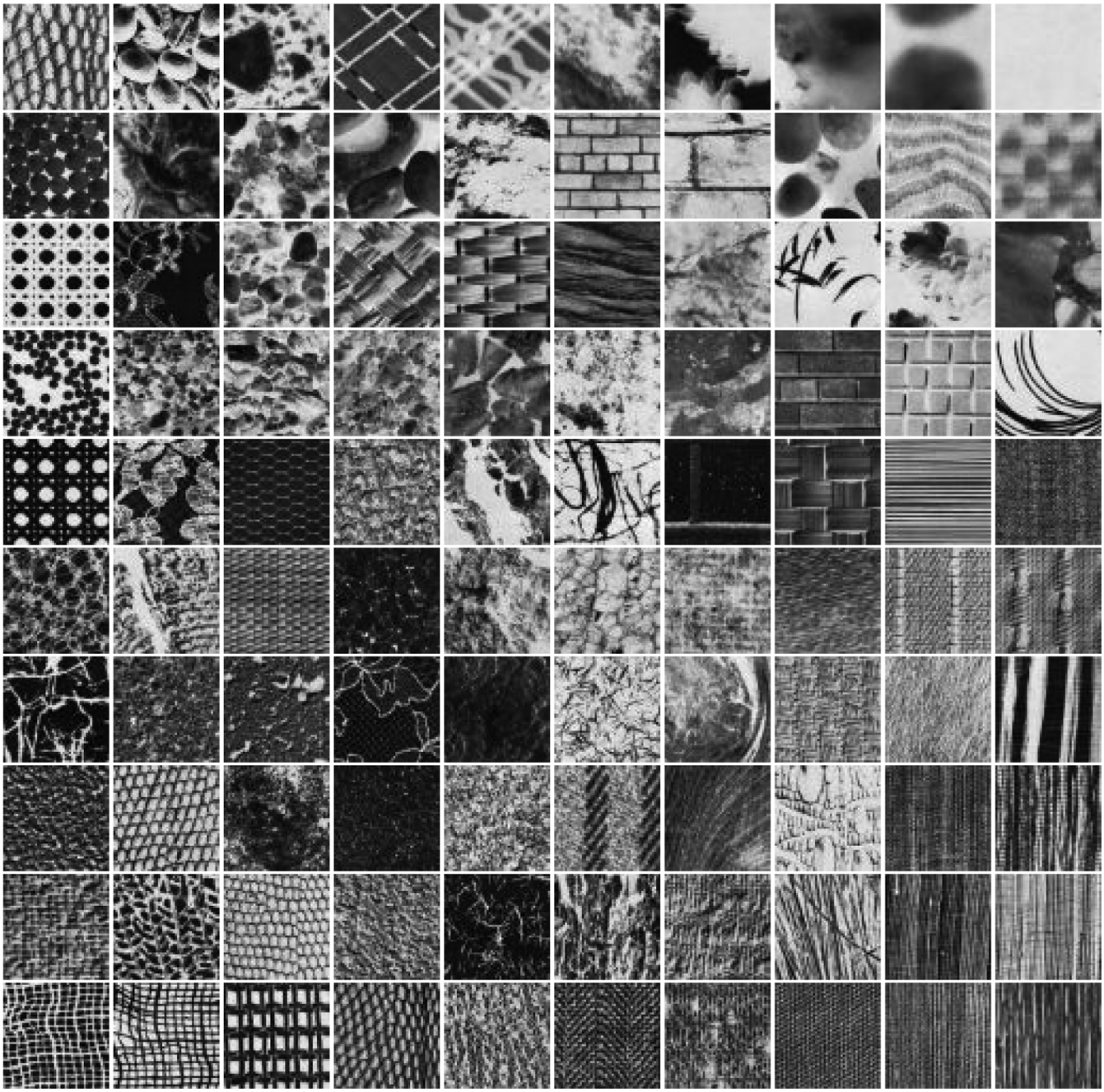

Fig. 15. An SSM organized set of 100 textures using gist vectors. Notice how regions of structurally similarity form with logical transitions between them.

embedded in 3D space. The "broken swiss" is similar but exhibits discontinuities. The "helix" and "twin peaks" datasets are non-linear. Multiple outcomes are demonstrated in Fig. 14. The most notable one is that none of the techniques are able to organize data in the "swiss roll" and "broken swiss" based on their underlying topological structures when the Euclidean distance between 3D vectors is used as the basis of organization. An overlap between the two layers of the 2D manifold manifests in the form of peppering (MDS, Sammon, and HexBoard) or tangled regions (SOM and SSM) due to the small Euclidean distance between the two layers.

One way to address this issue is to organize data using geodesic distance instead of the Euclidean distance. The geodesic distance between two data items is calculated along the shortest path between them within the k-nearest neighbor graph (k is set to 10). Hence, it measures the distance between data items along the 2D manifold and forces the rolls to unwind when they are organized. Nevertheless, the geodesic distances among different 3D vectors can only be represented using a distance matrix and so the SOM method cannot be applied due to the nature of the weight vector updates it requires. MDS, HexBoard, and the SSM with centroid representatives can work directly on the distance matrix and generate layouts that respect the underlying topological structures. In fact, running MDS over geodesic distance becomes the ISOMAP approach [35]. While the ISOMAP nicely unrolls the "swiss roll" dataset, noise

TABLE I
THE SEMANTIC RELATEDNESS AMONG DIFFERENT WIKIPEDIA ARTICLES RELATED TO THE QUERY "WASHINGTON". CELLS IN THE TABLE ARE SHADED BASED ON THEIR CORRESPONDING RELATEDNESS VALUES FOR BETTER VISUALIZATION. THE SHADING HELPS TO IDENTIFY A CLUSTER OF ARTICLES RELATED TO DENZEL WASHINGTON AND HIS MOVIES, BUT THE RELATIONS AMONG THE REST ARE UNCLEAR SINCE THEY ARE SOMEWHAT INTERCONNECTED

	TD	TGD	CF	FQC	DW	OT	IM	TBC	RC	GH	MV	MW	BF	GW	JM	CWP	AR	VF	FB	WC	USC	WM	WH	VC	GU	GWU	HU	WDC	Sea	MR	Ore	Mon	CR	ONP	CRi	WS
Training Day	1	0.71	0.7	0.58	0.62	0.68	0.67	0.62	0.67	0.64	0.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.32	0.1	0.09	0	0	0	0	0	0	0
The Great Debaters	0.71	1	0.62	0.68	0.75	0.74	0.71	0.66	0.52	0.47	0.52	0	0	0	0	0	0	0	0	0	0	0	0	0	0.31	0.22	0.45	0.19	0	0	0	0	0	0	0	0
Cry Freedom	0.7	0.62	1	0.64	0.68	0.64	0.59	0	0.5	0.62	0.5	0	0	0	0	0	0	0	0	0	0.36	0	0	0	0	0	0	0.15	0	0	0	0	0	0	0	0
For Queen and Country	0.58	0.68	0.64	1	0.67	0.73	0.63	0	0.48	0	0.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Denzel Washington	0.62	0.75	0.68	0.67	1	0.67	0.67	0.71	0.55	0	0.51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Out of Time	0.68	0.74	0.64	0.73	0.67	1	0.72	0.67	0.48	0	0.54	0	0.38	0	0	0	0	0	0	0	0	0	0.29	0	0.35	0	0	0	0.22	0	0	0	0	0	0	0
Inside Man	0.67	0.71	0.59	0.63	0.67	0.72	1	0.75	0.58	0.44	0.49	0	0	0	0	0	0.23	0	0	0	0	0	0.21	0	0	0	0	0	0.13	0	0	0	0	0	0	0
The Bone Collector	0.62	0.66	0	0	0.71	0.67	0.75	1	0.45	0.46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Russell Crowe	0.67	0.52	0.5	0.48	0.55	0.48	0.58	0.45	1	0.64	0	0	0.17	0	0	0	0	0	0	0	0.22	0.35	0.06	0	0.13	0.02	0	0.21	0.09	0	0.11	0.07	0	0	0	0
Gene Hackman	0.64	0.47	0.62	0	0	0	0.44	0.46	0.64	1	0	0	0.26	0	0	0	0	0	0	0	0.31	0.36	0.32	0	0.27	0.11	0	0.22	0.06	0	0.15	0	0	0	0	0
Mount Vernon	0.54	0.52	0.5	0.54	0.51	0.54	0.49	0	0	0	1	0.78	0.51	0.6	0.59	0	0.43	0.63	0.7	0	0.58	0.58	0.45	0.46	0.34	0	0.45	0.3	0.45	0.52	0.05	0.47	0.53	0	0.42	0.58
Martha Washington	0	0	0	0	0	0	0	0	0	0	0.78	1	0.54	0.69	0.61	0.53	0.46	0.64	0.57	0	0.54	0.44	0.43	0	0	0	0	0.27	0.08	0	0.16	0	0.31	0	0	0
Benjamin Franklin	0	0	0	0	0	0.38	0	0	0.17	0.26	0.51	0.54	1	0.57	0.65	0.48	0.62	0.56	0.37	0.41	0.47	0.29	0.31	0	0.28	0.25	0.18	0.38	0.14	0	0.15	0.22	0.09	0	0.29	0
George Washington	0	0	0	0	0	0	0	0	0	0	0.6	0.69	0.57	1	0.54	0.74	0.49	0.48	0	0	0.55	0	0.29	0	0	0.15	0	0.33	0.1	0	0.19	0.19	0.33	0	0	0
James Madison	0	0	0	0	0	0	0	0	0	0	0.59	0.61	0.65	0.54	1	0.46	0.58	0.54	0	0	0.6	0.49	0.48	0.51	0.51	0.36	0.54	0.5	0.11	0	0.57	0.7	0	0	0.34	0
Charles Willson Peale	0	0	0	0	0	0	0	0	0	0	0	0.53	0.48	0.74	0.46	1	0.45	0.6	0	0	0.35	0	0.28	0	0	0.18	0	0.14	0	0	0	0	0	0	0	0
American Revolution	0	0	0	0	0	0	0.23	0	0	0	0.43	0.46	0.62	0.49	0.58	0.45	1	0.54	0.39	0	0.41	0.32	0.35	0.19	0.26	0.24	0.24	0.44	0.05	0.11	0.18	0.19	0.07	0	0.15	0
Valley Forge	0	0	0	0	0	0	0	0	0	0	0.63	0.64	0.56	0.48	0.54	0.6	0.54	1	0.54	0.57	0.49	0.59	0.34	0	0	0	0.28	0.35	0	0.48	0.06	0.21	0	0.55	0	0.24
Fort Le Boeuf	0	0	0	0	0	0	0	0	0	0	0.7	0.57	0.37	0	0	0	0.39	0.54	1	0	0	0.52	0	0	0	0	0	0	0	0	0.22	0	0	0	0	0
Washington Circle	0	0	0	0	0	0	0	0	0	0	0	0	0.41	0	0	0	0	0.57	0	1	0.58	0.6	0.44	0	0.44	0.3	0	0.32	0	0	0	0	0	0.6	0	0
United States Capitol	0	0	0.36	0	0	0	0	0	0.22	0.31	0.58	0.54	0.47	0.55	0.6	0.35	0.41	0.49	0	0.58	1	0.69	0.61	0.5	0.44	0.17	0.41	0.52	0.27	0.41	0.31	0.36	0.15	0.31	0.25	0.3
Washington Monument	0	0	0	0	0	0	0	0	0.35	0.36	0.58	0.44	0.29	0	0.49	0	0.32	0.59	0.52	0.6	0.69	1	0.51	0.58	0.33	0.15	0.34	0.41	0.28	0.52	0.26	0.31	0	0.51	0.31	0.34
White House	0	0	0	0	0	0.29	0.21	0	0.06	0.32	0.45	0.43	0.31	0.29	0.48	0.28	0.35	0.34	0	0.44	0.61	0.51	1	0.33	0.45	0.14	0.41	0.56	0.27	0.17	0.24	0.25	0	0.17	0.11	0.25
Verizon Center	0	0	0	0	0	0	0	0	0	0	0.46	0	0	0	0.51	0	0.19	0	0	0	0.5	0.58	0.33	1	0.54	0	0.46	0.21	0.6	0	0.24	0.48	0	0	0	0.42
Georgetown University	0	0.31	0	0	0	0.35	0	0	0.13	0.27	0.34	0	0.28	0	0.51	0	0.26	0	0	0.44	0.44	0.33	0.45	0.54	1	0.34	0.52	0.36	0.42	0.46	0.45	0.47	0	0	0	0.14
George Washington U	0	0.22	0	0	0	0	0	0	0.02	0.11	0	0	0.25	0.15	0.17	0.18	0.24	0	0	0.3	0.17	0.15	0.14	0	0.34	1	0.24	0.21	0.14	0.14	0.07	0.1	0.1	0	0	0.19
Howard University	0.32	0.45	0	0	0	0	0	0	0	0	0.45	0	0.18	0	0.54	0	0.24	0.28	0	0	0.41	0.34	0.41	0.46	0.52	0.24	1	0.37	0.12	0.4	0.43	0.51	0	0	0	0.1
Washington DC	0.1	0.19	0.15	0	0	0	0	0	0.21	0.22	0.3	0.27	0.38	0.33	0.5	0.14	0.44	0.35	0	0.32	0.52	0.41	0.56	0.21	0.36	0.21	0.37	1	0.27	0.24	0.43	0.41	0.31	0.23	0.35	0.22
Seattle	0.09	0	0	0	0	0.22	0.13	0	0.09	0.06	0.45	0.08	0.14	0.1	0.11	0	0.05	0	0	0	0.27	0.28	0.27	0.6	0.42	0.32	0.12	0.27	1	0.47	0.45	0.41	0.46	0.34	0.42	0.61
Mount Rainier	0	0	0	0	0	0	0	0	0	0	0.52	0	0	0	0	0	0.11	0.48	0	0	0.41	0.52	0.17	0	0.46	0.14	0.4	0.24	0.47	1	0.43	0.39	0.7	0.63	0.56	0.61
Oregon	0	0	0	0	0	0	0	0	0.11	0.15	0.05	0.16	0.15	0.19	0.57	0	0.18	0.06	0.22	0	0.31	0.26	0.24	0.24	0.45	0.11	0.43	0.43	0.45	0.43	1	0.6	0.62	0.38	0.62	0.42
Montana	0	0	0	0	0	0	0	0	0.07	0	0.47	0	0.22	0.19	0.7	0	0.19	0.21	0	0	0.36	0.31	0.25	0.48	0.47	0.1	0.51	0.41	0.41	0.39	0.6	1	0.48	0.43	0.52	0.41
Cascade Range	0	0	0	0	0	0	0	0	0	0	0.53	0.31	0.09	0.33	0	0	0.07	0	0	0	0.15	0	0	0	0	0.1	0	0.31	0.46	0.7	0.62	0.48	1	0.54	0.74	0.56
Olympic National Park	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.55	0	0.6	0.31	0.51	0.17	0	0	0	0	0.23	0.34	0.63	0.38	0.43	0.54	1	0.52	0.38
Columbia River	0	0	0	0	0	0	0	0	0	0	0.42	0	0.29	0	0.34	0	0.15	0	0	0	0.25	0.31	0.11	0	0	0	0	0.35	0.42	0.56	0.62	0.52	0.74	0.52	1	0.55
Washington State	0	0	0	0	0	0	0	0	0	0	0.58	0	0	0	0	0	0	0.24	0	0	0.3	0.34	0.25	0.42	0.14	0.19	0.1	0.22	0.61	0.61	0.42	0.41	0.56	0.38	0.55	1

shows up in the "broken swiss" result due to the additional k-d tree alignment step. Since SSM directly organizes data items into a grid layout, its output for "broken swiss" is noticeably smoother.

For the non-linear "helix" and "twin peaks" datasets, on the other hand, all techniques generate logical results. Since the "helix" dataset is a 1D manifold, the MDS and Sammon approaches each map it to 1D curves in the raw organization (not shown) which results in blocky artifacts after the k-d tree alignment step converts it to a 2D grid. The HexBoard also shows artifacts after the grid conversion. In contrast, the SSM results using both mean and centroid are the smoothest ones with only a small number of isolated items; and the their correlation scores are the highest under the respective distance measures as well. The SSM with centroid also generates a topographically relevant layout for the twin peaks dataset with the two peaks being slightly separated, although its correlation score is not as high as the one obtained by ISOMAP.

Another observation is that the organization result depends highly on the distance measure used. For that reason it is a desirable feature that the centroid variant of the SSM algorithm is suited to using alternative distance measures without modification.

B. Testing on Different Multimedia Datasets

Next, we demonstrate the SSM's capability of handling different types of multimedia datasets. Previous research has shown that organizing images based on visual similarities can improve the users' browsing experience [27], [34]. Although a variety of techniques have been proposed for organizing images into complex structures, such as clusters or networks [9], popular search engines still present image search results in a grid layout due to its simple form.

In our preliminary work [31], we have illustrated how the SSM can be used to rearrange image search results based on visual similarities; while still maintaining the conventional 2D grid presentation. The GPU implementation presented in this paper dramatically improves the processing speed, allowing it to organize image search results online. This makes it a valuable tool for enhancing the current image search engines.

Besides organizing image search results, the SSM can also rearrange existing image collections. Fig. 15 shows the results obtained for 100 texture images from the well-known Brodatz collection (downloaded from http://www. ux.uis.no/~tranden/brodatz.html). In this case, we compute a 320-dimensional gist [21] feature vector for each image much like Joia *et al.* [14]. Gist vectors are low dimensional representations of scenes and are created by measuring edge responses from a series of convolutions on images using a filter at different orientations. The distances between gist vectors can be used to measure perceptual similarity [23]. Hence, the SSM result based on gist vectors arranges texture images based on overall edge and structure patterns.

Several clusters of distinct patterns can be observed from the organization in Fig. 15. For instance, we can find grid shaped textures at the bottom-left corner, vertical line shaped textures at the bottom right corner, and low frequency textures at the top-right corner. Nevertheless, the organization is not all ideal. The upper middle region, for example, is less consistent. This region's heterogeneity is likely due to the combination of ambiguous gist vectors and the compromise that the SSM made in order to place all textures into the constrained layout.

The input to the SSM in the above experiment is high-dimensional vectors, i.e., 320-dimensional gist feature vector. The

TABLE II
THE ORGANIZATION RESULT FOR DIFFERENT WIKIPEDIA ARTICLES BASED ON THEIR RELATEDNESS. FOR BETTER VISUALIZATION, CELLS ARE SHADED BASED ON THE CLUSTER THAT THE ARTICLE BELONGS TO. NOTE THAT THE CLUSTER INFORMATION IS NOT AVAILABLE TO THE SSM ALGORITHM, YET IT WAS ABLE TO GROUP ARTICLES IN THE SAME CLUSTER TOGETHER

Gene Hackman	Russell Crowe	**Denzel Washington**	The Great Debaters	For Queen & Country	Training Day
Cry Freedom	Inside Man	The Bone Collector	Out of Time	Mount Vernon	Seattle
Fort Le Boeuf	Washington Circle	White House	Washington Monument	Mount Rainier	Cascade Range
Valley Forge	Martha Washington	United States Capitol	**Washington DC**	Olympic National Park	Columbia River
Charles Willson Peale	**George Washington**	James Madison	Georgetown University	Oregon	**Washington State**
American Revolution	Benjamin Franklin	George Wash-ington U.	Howard University	Montana	Verizon Center

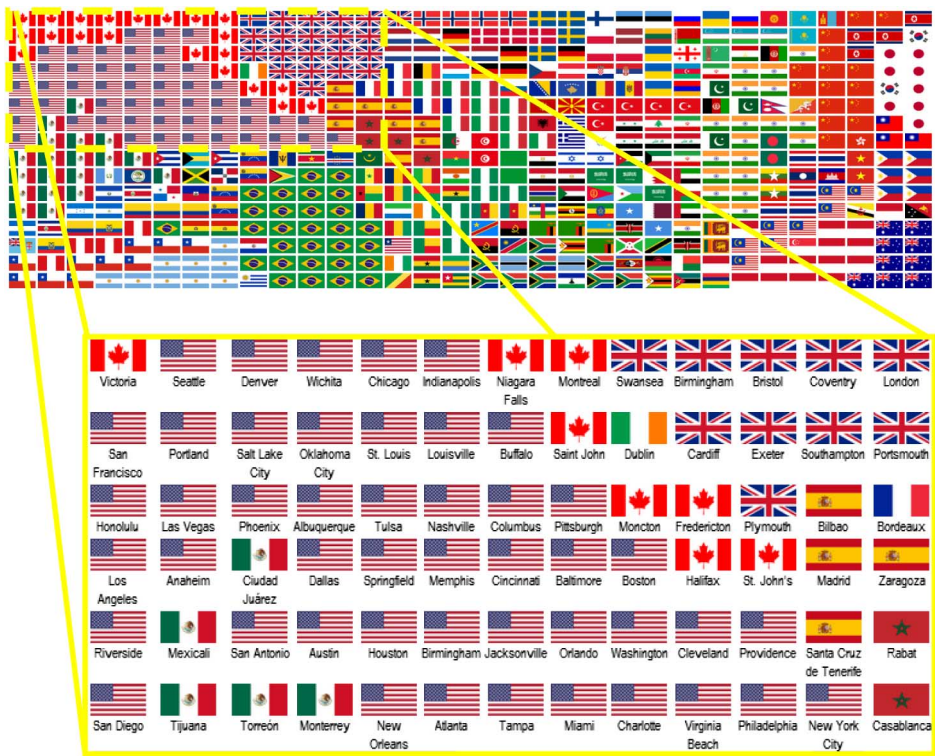

Fig. 16. The world's largest 512 cities arranged by the SSM. The top shows the country flags of the cities of the entire 32×16 organization followed by a zoomed in view, in which the city names are marked.

next experiment uses the SSM to organize nominal-valued data, which are a set of Wikipedia articles retrieved using the query "Washington". Due to the ambiguousness of the query, a diverse set of articles is retrieved, ranging from persons, to movies, to places. The semantic relatedness between any two articles is computed using Wiki Miner [19]. As shown in Table I, it is difficult to grasp relatedness information from the table directly. Hence, an effective organization method is warranted. This is a typical nominal dataset, where means cannot be computed. To organize this data, we negate the semantic relatedness values in Table I and use them as dissimilarity. The organizational result in Table II shows the relation between things corresponding to Denzel Washington and his movies, Washington, DC and its landmarks, Washington State and its neighboring states, as well as George Washington and the persons and places related to him. It is also worth noting that the Washington, DC cluster and the George Washington cluster are placed adjacent to each other because of their relatively high relatedness. The same phenomenon can be observed between Washington, DC and Washington State.

The last dataset tested contains the major cities of the world, which are represented using a data structure containing both real and nominal values. Each city data item A carries three properties: its latitude A_{lat}, longitude A_{long}, and the name of its nation A_{nation}. The objective is to place cities based on geological proximity, while at the same time encouraging cities from the same nation to be clustered together.

To achieve this goal, we define the dissimilarity between two cities A and B to be based on both the great-circle distance between the respective latitudes and longitudes, and their country membership:

$$\delta(A,B) = (A_{nation} == B_{nation} ? \alpha : 0) + (1-\alpha) \left(2r\sin^{-1}\sqrt{\begin{array}{l}\sin^2\frac{B_{lat}-A_{lat}}{2} + \\ \cos A_{lat}\cos B_{lat}\sin^2\frac{B_{long}-A_{long}}{2}\end{array}} \right) \tag{10}$$

where the first term represents the country influence, the second computes the great-circle distance between the two cities (r

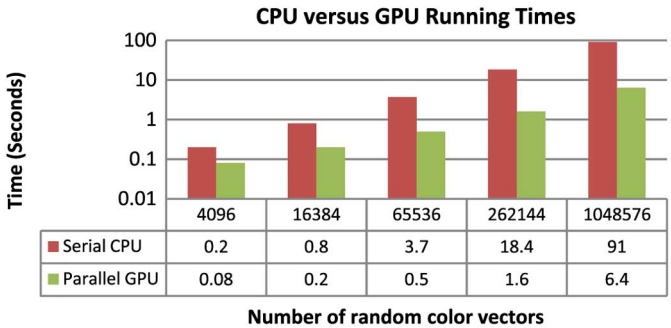

Fig. 17. The running times of the serial implementations versus the parallel GPU implementation on Lab color vector datasets of varying size. The parallel GPU implementation can arrange a million items in 6.5 seconds.

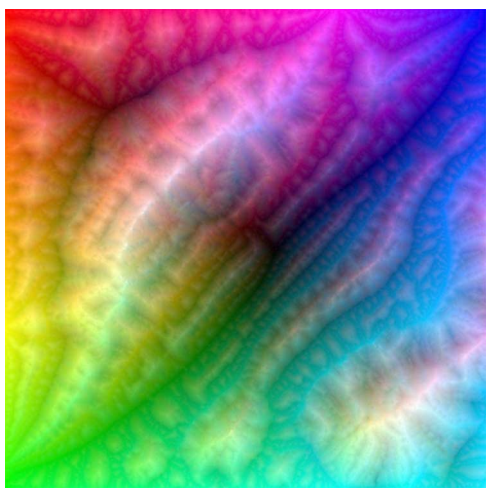

Fig. 18. The result for organizing 1 million Lab color vectors into a 1024×1024 grid. The organization reveals that fractal-like structures are automatically formed to flatten the 3D color space onto a 2D plane.

being the earth's radius), and the α balances between the two terms and is set to 0.5 in our experiment.

Here boundary conditions have been put in so that the SSM organization not only respects city to city distances but also follows the convention of the world map. For example, the constraints across the northern border of the map vary from 90°N, 180°W to 90°N, 0° to 90°N,180°E for top left to top right corner. The other borders are given suitable constraints based on their relative positions as well. The rest of the layout is then guided by the dissimilarity between cities; see Fig. 16. Such a layout is useful for displaying various information for these cities, such as weather shown in Fig. 1, on a public billboard. Note that the dissimilarity measure given in Equation (10) is merely a proof of concept; presumably encoding different information about the relationships between places (trade, lending, conflict, aid, etc.) could produce other interesting results describing political landscapes.

C. Benchmarks on Processing Speed

As a final experiment, we measure the processing speed of both serial and parallel implementations of the SSM algorithm over Lab color vector datasets of different sizes. The result for 4096 color vectors can be seen in Fig. 13, whereas the layout for one million vectors is shown in Fig. 18.

The serial version is implemented using a single-threaded Java program, whereas in the parallel version several core functions are replaced with OpenCL kernel functions, which are invoked through JOCL bindings. To make the time measurements comparable across different implementations and datasets, we force the algorithm to go through 5 iterations per stage, regardless whether the process converges or not. The CPU used is an Intel Xeon E5540 running at 2.5 GHz. The GPU used is a NVIDIA GeForce GTX 480 which has 480 streaming multiprocessors across 15 compute units running at 1.4 GHz. It is clear from the timing in Fig. 17 that the parallelism of the SSM produces an impressive speedup.

VI. Conclusion

A novel algorithm for organizing and visualizing multimedia data is presented in this paper. The algorithm borrows many ideas and features from established dimension reduction techniques, yet it has a different objective than these techniques. Instead of solving the continuous optimizing problem as other dimension reduction approaches do, the SSM transforms it into a discrete labeling problem. As a result, it can organize a set of data into a structured layout without overlap, providing a simple presentation directly, making it suitable for presenting multimedia data items on websites; see Fig. 1. The SSM is flexible in terms of input. It can organize numeric data like all its counterparts or non-numeric data using a provided dissimilarity matrix like MDS. The output style of the SSM does bare resemblances to the SOM in that it is a structured grid. However, a key difference is that the SOM organizes data indirectly through its weight vectors while the SSM holds and works on the actual data directly making it possible to organize non-numeric data without computing a contrived numeric average between items of the data. Given any data for which pairwise dissimilarity can be defined, the SSM rearranges that input data in a map to produce near optimal data placement such that the distance between any pair of data items correlates positively with the dissimilarity between them.

Experiments on different types of data show that the SSM can be applied to a variety of applications, ranging from rearranging textures based on visual similarities to visualizing semantic relatedness between articles to the generation of alternative yet intuitive world maps. The results clearly demonstrate that the SSM works toward the goal of producing a topology preserving organization for the input data within the confines of the output structure that is given. The SSM algorithm is fully parallel by design and as such can run efficiently on parallel hardware. Using a GPU we can organize tens of thousands of data in a fraction of a second and over a million in close to six seconds. These are very reasonable speeds for interactive applications.

When used for visualizing data, the SSM presents all data items in an occlusion free manner. In some cases (see Fig. 18) presenting millions of data items gives users a good visualization of the dataset, in many others presenting too much data at once can overwhelm the users. In these cases, the multi-resolution browsing scheme proposed for SOM [30] can be adopted here to allow users to explore the data organization in a coarse-to-fine manner using pan and zoom operations.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive and valuable comments. Thanks also go to Roberto De Phino, M. C. F. de Oliveira, and A. de Andrade Lopes, for providing us with their HexBoard implementation. This research is supported by the NSERC and Memorial University of Newfoundland.

REFERENCES

[1] Algorithmics Group, "MDSJ: Java library for multidimensional scaling (version 0.2)," 2009.
[2] I. Borg and P. Groenen, *Modern multidimensional scaling: Theory and applications*, 2nd ed. New York, NY, USA: Springer-Verlag, 2005.
[3] T. T. Chen and L. C. Hsieh, "The visualization of relatedness," in *Proc. Int. Conf. Inf. Visualisation*, 2008, pp. 415–420.
[4] M. Ciura, "Best increments for the average case of shellsort," in *Proc. Int. Symp. Fundamentals of Computation Theory*, 2001, pp. 106–117.
[5] R. D. de Pinho, M. C. F. de Oliveira, and A. de Andrade Lopes, "An incremental space to visualize dynamic data sets," *Multimedia Tools and Applications*, vol. 50, no. 3, pp. 533–562, 2010.
[6] G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1999.
[7] T. Dwyer, K. Marriott, and P. J. Stuckey, "Fast node overlap removal," in *Proc. Graph Drawing*, 2006, pp. 153–164.
[8] E. Gomez-Nieto, W. Casaca, L. G. Nonato, and G. Taubin, "Mixed integer optimization for layout arrangement," in *Proc. Sibgrapi: Conf. Graph. Imaging and Vision*, 2013.
[9] D. Heesch, "A survey of browsing models for content based image retrieval," *Multimedia Tools and Applications*, vol. 42, no. 2, pp. 261–284, 2008.
[10] E. Hoque, O. Hoeber, and M. Gong, "CIDER: Concept-based image diversification, exploration, and retrieval," *Inf. Process. & Management*, vol. 49, no. 5, pp. 1122–1138, 2013.
[11] E. Hoque, G. Strong, O. Hoeber, and M. Gong, "Conceptual query expansion and visual search results exploration for Web image retrieval," in *Proc. Atlantic Web Intell. Conf.*, 2011, pp. 73–82.
[12] S. Hotton and J. Yoshimi, HiSee. ver. 1.0.0, 2004 [Online]. Available: http://hisee.sourceforge.net/
[13] R. E. Jensen, *Self-sorting map collarboration*. Trondheim, Norway: , 2012.
[14] P. Joia, F. V. Paulovich, D. Coimbra, J. A. Cuminato, and L. G. Nonato, "Local affine multidimensional projection," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2563–2571, 2011.
[15] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.
[16] S. Kaski and J. Peltonen, "Dimensionality reduction for data visualization," *IEEE Signal Process. Mag.*, vol. 28, no. 2, pp. 100–104, 2011.
[17] T. Kohonen, *Self-Organization Maps*. Berlin, Germany: Springer-Verlag, 1995.
[18] T. Kohonen, S. Kaski, K. Lagus, J. Salojarvi, V. Paatero, and A. Saarela, "Self organization of a massive document collection," *IEEE Trans. Neural Networks*, vol. 11, no. 3, pp. 574–585, 2000.
[19] D. Milne and I. H. Witten, "An effective, low-cost measure of semantic relatedness obtained from Wikipedia links," in *Proc. AAAI Workshop on Wikipedia and Artif. Intell.*, 2008.
[20] A. Morrison, G. Ross, and M. Chalmers, "Fast multidimensional scaling through sampling, springs and interpolation," *Inf. Visualization*, vol. 2, no. 1, pp. 68–77, 2003.
[21] F. V. Paulovich, D. Eler, J. Poco, C. P. Botha, R. Minghim, and L. Nonato, "Piece wise Laplacian-based projection for interactive data exploration and organization," in *Proc. Comput. Graphics Forum*, 2011, pp. 1091–1100.
[22] F. V. Paulovich, C. T. Silva, and L. G. Nonato, "Two-phase mapping for projecting massive data sets," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 6, pp. 1281–1290, Jun. 2010.
[23] F. V. Paulovich, C. T. Silva, and L. G. Nonato, "User-centered multidimensional projection techniques," *Computing in Science & Eng.*, vol. 14, no. 4, pp. 74–81, 2012.
[24] R. Pinho and M. C. F. de Oliveira, "Hexboard: Conveying pairwise similarity in an incremental visualization space," in *Proc. 13th Int. Conf. Inf. Visualisation*, , 2009, pp. 32–37.
[25] R. Pinho, M. C. F. de Oliveira, and A. de A Lopes, "Incremental board: A grid-based space for visualizing dynamic data sets," in *Proc. ACM Symp. Appl. Computing*, 2009, pp. 1757–1764.
[26] F. H. Post, G. M. Nielson, and G.-P. Bonneau, *Data Visualization: The State of the Art*. Berlin, Germany: Springer-Verlag, 2002.
[27] K. Rodden, W. Basalaj, D. Sinclair, and K. Wood, "Does organisation by similarity assist image browsing?," in *Proc. SIGCHI Conf. Human Factors in Computing Syst.*, 2001, pp. 190–197.
[28] J. W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Trans. Computing*, vol. 18, no. 5, pp. 401–409, 1969.
[29] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen, "Rolled-out wordles: A heuristic method for overlap removal of 2D data representatives," in *Proc. Comput. Graphics Forum*, 2012, pp. 1135–1144.
[30] G. Strong and M. Gong, "Browsing a large collection of community photos based on similarity on GPU," in *Proc. Int. Symp. Vis. Computing*, 2008, pp. 390–399.
[31] G. Strong and M. Gong, "Data organization and visualization using self-sorting map," in *Proc. Graphics Interface*, 2011, pp. 199–206.
[32] G. Strong and M. Gong, "Organizing and browsing photos using different feature vectors and their evaluations," in *Proc. Int. Conf. Image and Video Retrieval*, 2009, pp. 1–8.
[33] G. Strong, O. Hoeber, and M. Gong, "Visual image browsing and exploration (vibe): User evaluations of image search tasks," in *Proc. Int. Conf. Active Media Technol.*, 2010, pp. 424–435.
[34] G. Strong, E. Hoque, M. Gong, and O. Hoeber, "Organizing and browsing image search results based on conceptual and visual similarities," in *Proc. Int. Symp. Vis. Computing*, 2010, pp. 481–490.
[35] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5550, pp. 2319–2323, 2000.
[36] A. Tikhonova and K.-L. Ma, "A scalable parallel force-directed graph layout algorithm," in *Proc. Eurographics Parallel Graphics and Vis. Symp.*, 2008, pp. 25–32.
[37] L. Van der Maaten, E. Postma, and H. Van den Herik, "Dimensionality reduction: A comparative review," *J. Mach. Learning Res.*, vol. 10, no. 1, pp. 1–41, 2009.
[38] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski, "Information retrieval perspective to nonlinear dimensionality reduction for data visualization," *J. Mach. Learning Res.*, vol. 11, pp. 451–490, 2010, no. Feb.
[39] J. Zhang, C. Chen, and J. Li, "Visualizing the intellectual structure with paper-reference Matrices," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 6, pp. 1153–1160, 2009.

Grant Strong currently works as a Software Engineer for Google in California. He did his graduate studies at the Memorial University of Newfoundland (MUN) with degrees in Computer Science and Education. He received his M.Sc. degree in 1999 and later PhD in 2013, both in Computer Science from MUN.

His interests lie in the fields of visual computing, data organization, and augmented reality. Over the course of his graduate career he has worked in St. John's, Canada at MUN, Trondheim, Norway at NTNU, and New York at Google. During that time his work has been presented internationally and published in 7 referred conference papers and 3 referred journal articles.

Minglun Gong (M'03) received the M.Sc. degree in Computer Science from the Tsinghua University, China, in 1997 and the Ph.D. degree from the University of Alberta, Canada, in 2003.

He is currently an associate professor at the Memorial University in Canada. His research interests cover various topics in the broad area of visual computing (including computer graphics, computer vision, visualization, image processing, and pattern recognition). So far, he has published over 80 technical papers in refereed journals and conference proceedings and submitted 4 patent applications. He has served as program committee member for top-tier conferences, such as ICCV and CVPR; and he was the recipient of the Izaak Walton Killam Memorial Award and the CFI New Opportunity Award.