VRGrid: Efficient Transformation of 2D Data into Pixel Grid Layout

Adrien Halnaut, Romain Giot, Romain Bourqui, David Auber
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

{adrien.halnaut,romain.giot,romain.bourqui,david.auber}@labri.fr

***Abstract*—Projecting a set of n points on a grid of size $\sqrt{n} \times \sqrt{n}$ provides the best possible information density in two dimensions without overlap. We leverage the Voronoi Relaxation method to devise a novel and versatile post-processing algorithm called VRGrid: it enables the arrangement of any 2D data on a grid while preserving its initial positions. We apply VRGrid to generate compact and overlap-free visualization of popular and overlap-prone projection methods (*e.g.*, t-SNE). We prove that our method complexity is $O\left(\sqrt{n}.i.n.log(n)\right)$, with i a determined maximum number of iterations and n the input dataset size. It is thus usable for visualization of several thousands of points. We evaluate VRGrid's efficiency with several metrics: distance preservation (DP), neighborhood preservation (NP), pairwise relative positioning preservation (RPP) and global positioning preservation (GPP). We benchmark VRGrid against two state-of-the-art methods: Self-Sorting Maps (SSM) and Distance-preserving Grid (DGrid). VRGrid outperforms these two methods, given enough iterations, on DP, RPP and GPP which we identify to be the key metrics to preserve the positions of the original set of points.**

***Index Terms*—visualization, compact visualization, evaluation**

I. Introduction

Data visualization is an efficient way to analyze and communicate numerical data [1] that are intractable in tabular or textual formats. Such numerical data are varying in size (*i.e.*, number of elements in the dataset) and dimensionality (*i.e.*, number of features for each element). Many visualization techniques [2] have been designed to get insights from high-dimensional datasets, for instance, histograms, scatter plots, etc... However, all these techniques are constrained by the resolution of the end-user screen (*i.e.*, number of available pixels). This constraint makes visualization of high-dimensional data challenging, especially when the number of dimensions is higher than the number of available visual attributes [3]. Tackling these challenges is crucial as usage of high-dimensional data is now common in many research and industrial fields.

High dimensional data visualization is usually done by reducing the number of dimensions until it fits into the number of available visual attributes. Various dimension reduction techniques exist for different purposes [4]; they focus on optimizing different criteria, and their visualization method is usually a scatter plot. High-dimensional data visualization techniques are efficient in many fields [5], but the larger the dataset is, the larger the drawing space needs to be [6]. Popular modern high-dimension reduction techniques such as t-SNE [7] or UMAP [8] produce non-uniform data projections that group together similar elements of the dataset; as the size of the high-dimensional dataset increases, those groups are getting denser, and the visual occlusion of individual elements is globally increased. If not handled properly, this visual occlusion results in wrong assumptions about the dataset's nature. For example, in Figure 1, MNIST dataset (third column, top row) is visualized using UMAP dimension reduction technique, and each element is colorized according to its class. The yellow cluster on the top-left part of the projection seems to be composed of only about 4 outliers, while in reality there are more than 20 outliers hidden behind other elements which are properly displayed with VRGrid (third column, second and third rows).

Several methods exist to increase the perceptual scalability of dimension reduction methods. In this paper, we make the choice to design a pixel-oriented method that relies on a grid layout. Pixel-oriented methods aim to provide informative data visualization while prioritizing perceptual scalability over accurate spatial fidelity. This perceptual scalability is achieved by removing visual occlusion and maximizing drawing space usage, resulting in a compact visualization technique of data. Grid-layout arrangements are a subset of pixel-oriented visualization methods which are scaled to the pixel size, free of any overlapping data. State-of-the-art method Self-Sorting Map [9] provides compact results with good information retention compared to regular non-uniform data projections. The method consists in populating a grid with high-dimensional data and performing permutations to optimize a correlation metric based on the dissimilarity between elements in both the high-dimensional space and in the grid's 2D space. Despite the good results, focusing on a single comparison criterion seems limited compared to the numerous regular data projection methods existing in the dimension reduction field. State-of-the-art method DGrid [10], [11] is a two-step process that takes as input an already projected and performs arrangements on projected elements. Those arrangements are processed by first partitioning the drawing space and then performing arrangements locally. It prioritizes neighborhood conservation and is designed to scale easily to larger datasets.

We propose a novel method that produces a grid arrangement of any 2D data, Voronoi Relaxation Grid (VRGrid). Compared to other arrangement methods, VRGrid takes 2D data as input (which may be projected high-dimensional data) and makes

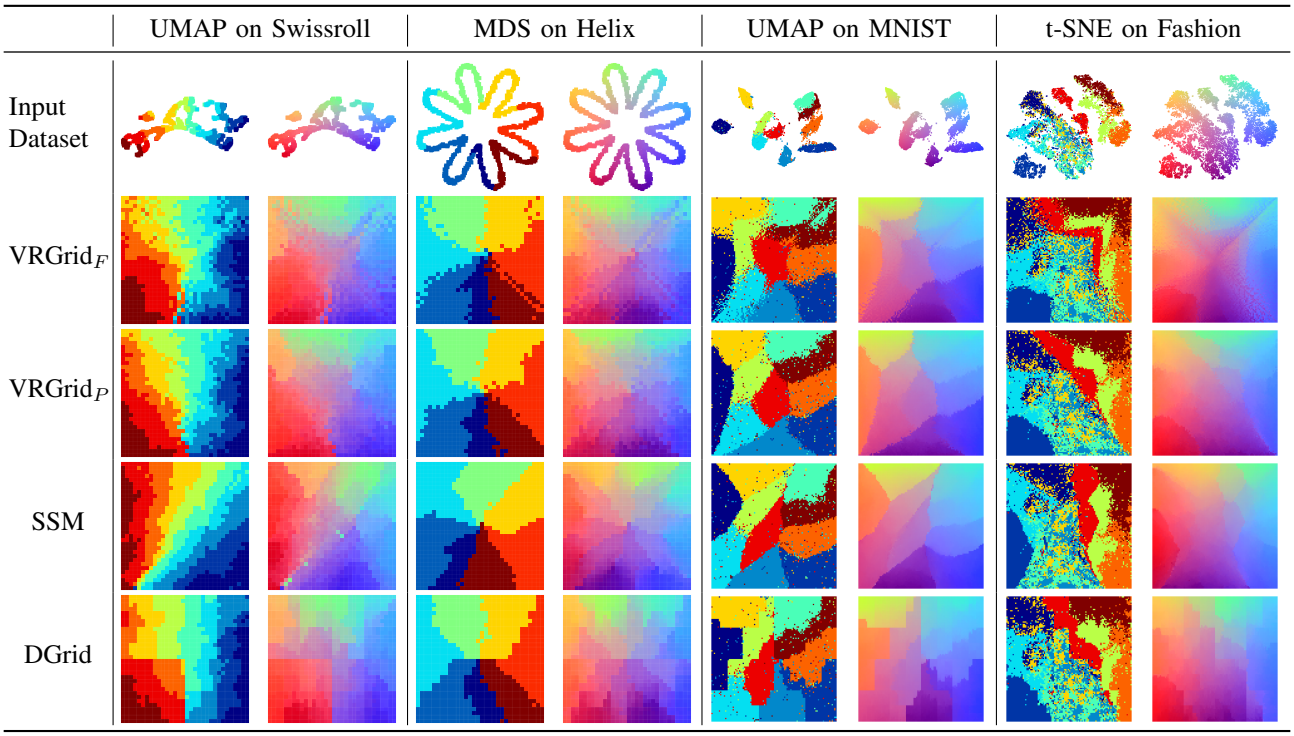


Fig. 1. Sample of processed scenario for all four methods. Each presented scenario is shown using two colorization rules. The left column of each scenario has elements colorized according to their class in the original dataset. The right column of each scenario has elements colorized according to their 2D position before being processed by the methods. In both cases, elements are keeping their color before (first line) and after processing.

use of a relaxation algorithm to uniformly scatter the 2D points into a finite space. Points are progressively assigned into grid cells while respecting the configuration computed by the relaxation method. The relaxation is applied multiple times to progressively refine the placements of points that are still unassigned to grid cells. This process is done until all points are assigned to a grid cell. VRGrid can be used as a complementary visualization method of already projected high-dimensional data.

In this paper, the relaxation method used by VRGrid is Lloyd's algorithm [12], which is used to compute Centroid Voronoi Tessellations [13], [14] (CVT) configurations that is helpful in uniformly scattering a set of coordinates in a 2D space. We conduct a quality evaluation of VRGrid arrangements using various datasets, against state-of-the-art methods. The contributions of this paper are:

- a novel iterative method able to arrange any set of 2D points into a grid with minimal deformation
- a public implementation of the algorithm
- an exhaustive comparison between VRGrid and state-of-the-art methods SSM and DGrid

The outline of this paper is as follows. Section II presents state-of-the-art methods to produce grid-based arrangements. Section III presents the VRGrid algorithm, its limitations and an analysis of its computation cost. Section IV presents how the evaluation protocol has been conducted and Section V its results. Finally, Section VI summarizes the paper and highlights how VRGrid can be improved.

II. Related Works

Regular data projections are efficient in representing data on a screen, but their construction requires empty drawing space to represent differences between two elements and often results in overlapping data when the drawing space is too small. Various techniques exist to project methods that aim to optimize readability over accurate distance encoding in data representation, and are presented in this section.

A. General Space-Filling Methods

Space-filling visualization is a way to improve this drawing space usage by encoding input data into a structure that fills a squared drawing space. **Fractal Curves** are an efficient way to represent data organization. It consists in ordering high-dimensional data along a fractal curve using a dissimilarity metric. Fractal curves such as Hilbert's [15] are designed to preserve the proximity of their elements once they are drawn in a 2D space. Keim's work on pixel-oriented visualization [16] shows that this technique is used in several application domains. Those fractal curves are not limited to orthogonal structures; GosperMap [17] makes use of Gosper curves to present hierarchical data in an undetermined visual structure, resulting in a geographic map-looking visualization. However, while being efficient in their application domain, it is necessary to reduce the input data to a 1D-space (*i.e.*, the order on the fractal curve), which implies a large information loss in the case of complex data.

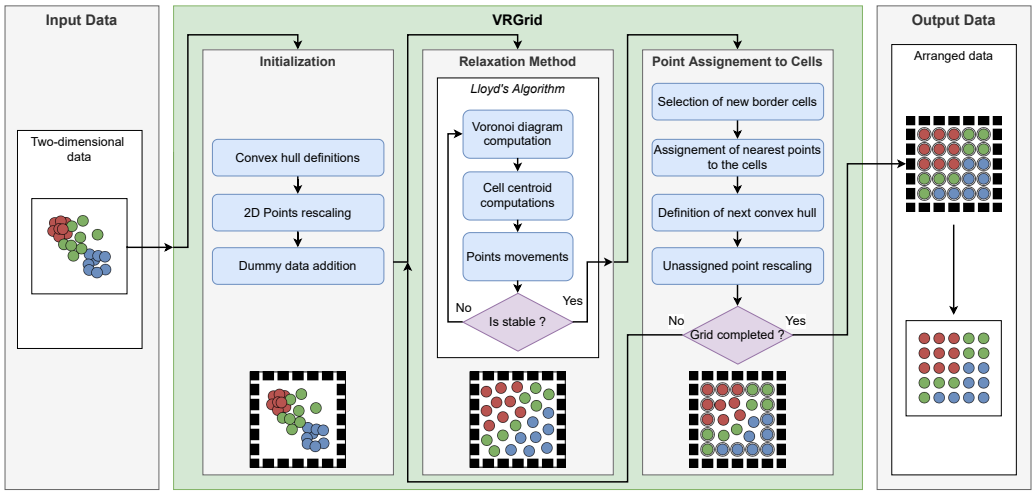


Fig. 2. VRGrid usage presentation. It consumes a set of 2D points and outputs a grid-based arrangement of it. It first initializes the grid structure and prepares the data to be processed. Then, VRGrid iteratively scatter data using a relaxation method and assign points in the current border cells until all points are assigned.

B. Grid Arrangement Methods

Grid-based projection methods can be categorized into two subsets, the first one projects data directly into a grid layout starting from high-dimensional space, while the other makes use of an intermediate dimensional reduction method before organizing the lower-dimensional result into a grid.

IsoMatch [18] makes use of the IsoMap [19] dimension reduction technique to project high dimensional data into a 2D space, followed by the Hungarian algorithm [20] to fit projected data into a specific structure (usually a grid). The Hungarian algorithm performs combination optimizations over an assignment problem. In compact data projection, the assignment problem is presented as positioning elements into a grid with each possible assignment associated with a cost function, usually representative of the data deformation it would create. While the Hungarian algorithm is designed to find the best possible configuration, its high computational cost of $O(n^4)$, with n being the dataset's size, makes it hardly usable for large dataset, and by extension the IsoMatch method itself.

VRGrid uses a similar approach as it consumes projected data as an input and performs arrangement from it to build a grid. However, VRGrid's computational complexity can be adapted to scale to large datasets at the cost of arrangement quality, as explained in Section III-B.

DGrid (Distance-preserving Grid) [10], [11] is a grid arrangement method designed to be a post-processing step of reduction dimension method. It consumes projected data positions as input and partitions the drawing space according to their sorting order on both x and y axes. Partitions are then progressively split in half until they can fit in a predetermined grid cell. This method is shown to be very efficient in both neighborhood preservation and processing time, and its complexity of $O(N.logN)$ makes it scalable to large problems. However, due to the partitioning strategy, deformation of data topology can occur on unevenly scattered data (*e.g.*, t-SNE).

In comparison, VRGrid strategy is not based on partitioning data and thus its grid arrangement does not produce such deformations, but at the cost of a higher computational cost.

Self-Sorting Map (SSM) [9] takes high-dimensional data and arranges them onto a grid such that the distance between arranged elements represents dissimilarity between them. Elements are first randomly disposed into a grid, then smartly swapped to maximize the correlation between distances and a given dissimilarity metric. SSM progressively split the grid and manages to produce structured layouts in a relatively short time frame with a given complexity of $O(L(logN)^2)$ and can be lowered using parallel computing. This method is proven to be efficient, but only dissimilarity information is used to arrange the input data. Topology data such as the positions of the input dataset are unused. VRGrid strongly differs from this approach as the input data coordinates are taken into account during the grid construction, and relies on already projected data to work.

III. Voronoi Relaxation Grid

The VRGrid method (summarized in Figure 2) aims at computing a square grid-layout arrangement for a given dataset P (but can be easily generalized for rectangles). There is no restriction on the nature of the dataset if each element of the dataset is a 2D coordinate. VRGrid relies on the application of a relaxation algorithm able to evenly scatter 2D points into a convex and bounded 2D space. It first initializes a structure for the 2D points of the input dataset to be placed into, then, for every iteration of the method, applies the relaxation algorithm and iteratively assigns scattered 2D points to the grid cells. Only a subset of points is assigned at each VRGrid iteration, starting from the edges of the grid toward its center, to prevent distortion of the configurations computed by the relaxation method. The relaxation method is then reapplied on unassigned points in a smaller space and the process goes on until all

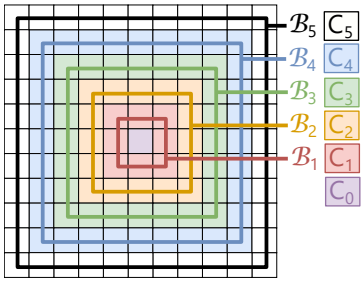
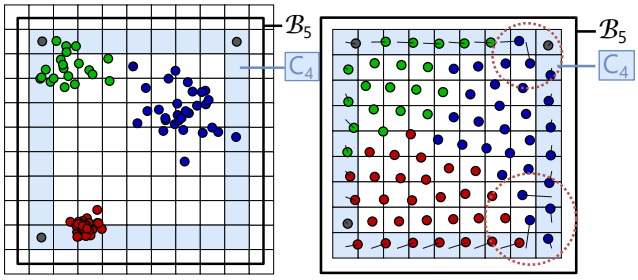


Fig. 3. Definition of convex hulls ($\mathcal{B}_1$ to $\mathcal{B}_5$) and associated delimiting cells (C_0 to C_5, colorized according to their appropriated convex hull) for a 11×11 sized grid used for a 9×9 sized dataset.



(a) VRGrid initialization (b) Result of Lloyd's algorithm after 223 iterations

Fig. 4. Overview of VRGrid's first iteration on a sample dataset. In (a), current convex hull is $\mathcal{B}_5$ and C_4 (light blue cells) is the first set of cells to be assigned P points to, as defined in Figure 3. Points have been rescaled to fit into the convex hull and $9 \times 9 - 78 = 3$ dummy points (grey circles) have been placed in C_4 to use all grid's cells at the end of the algorithm. In (b), segments represent the nearest point of P_0 for every cell of C_4. Red dashed circle highlights assignments that may seem unintuitive due to the assignment order. This is due to how the order of assignments is computed which results in farther candidates as the assignments go.

points are assigned to a grid cell. This section describes each step of VRGrid's process and explains the design choices.

A. Algorithm

Initialization. For a given dataset P composed of n 2D points, we note G the grid structure where P will be arranged. G is a square grid composed of $(\lceil\sqrt{n}\rceil + 2) \times (\lceil\sqrt{n}\rceil + 2)$ cells: the $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ cells where P points will be assigned during the VRGrid application, and $4 \times (\lceil\sqrt{n}\rceil + 1)$ additional cells around it to facilitate the algorithm execution. However, these cells will not contain points. We note $\mathcal{B}_0, \mathcal{B}_1, ...\mathcal{B}_k$ the convex hulls centered around the middle of G, with $k = \lceil\sqrt{n}/2\rceil + 1$. Each convex hull $\mathcal{B}_i$ corresponds to a square with a width of w_i cells such as $w_i = (2i - 1)$ if $\lceil\sqrt{n} + 1\rceil$ is odd and $w_i = 2(i + 1)$ otherwise.

Figure 3 illustrates the construction of the borders of the convex hulls and their appropriate representing cells for an 11×11 sized grid. We note C_i the set of cells of G that represents $\mathcal{B}_i$ for each convex hull. If $\lceil\sqrt{n} + 1\rceil$ is odd, C_0 is composed of the middle cell of G. These sets are chosen in a way that at the i-th iteration of VRGrid, 2D points are relaxed inside the space delimited by $\mathcal{B}_{k-i}$ and are assigned to the cells in the C_{k-i-1} set.

Next, 2D coordinates of the points of P are rescaled to fit into $\mathcal{B}_k$; this ensures that there is enough grid cells inside $\mathcal{B}_k$ for the 2D points to be assigned to: $|C_{k-1} \cup C_{k-2} \cup ... \cup C_0| \geq |P|$. The last initialization step of VRGrid aims to handle cases where $|P| < \lceil\sqrt{n}\rceil^2$ where there are more grid cells than points to assign. This configuration would result in empty cells and because of VRGrid's iterative nature, these empty cells would be located toward the middle of the grid and hinder its visualization. To prevent this configuration from happening, we add "dummy" data into P. These "dummy" points are processed in the same way as the rest of P but are removed from the result to not be confused with the actual data. We chose to uniformly place them in B_{k-1}, the relaxation algorithm can still change their coordinates in next step. To illustrate an iteration of VRGrid step by step, Figure 4a shows complete initialization of VRGrid on a randomly generated dataset of 78 points, distributed as three clusters.

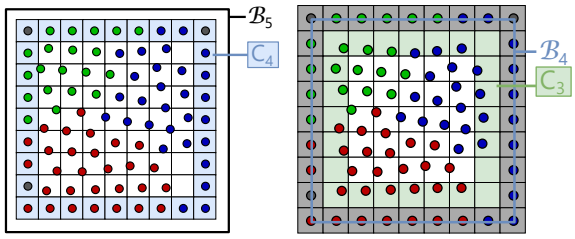
Relaxation Method. VRGrid makes use of relaxation algorithm to scatter 2D points in a finite and convex space. At the i-th iteration of VRGrid starting with $i = 0$, this space is defined by the $\mathcal{B}_{k-i}$ convex hull, and unassigned points P_i are processed by the relaxation algorithm such as $P_i = P_{i-1} \backslash C_{k-i}$ with $P_0 = P$. The relaxation algorithm needs to evenly scatter P_i points toward the cells in C_{i-1}.

In this paper, we chose to use Lloyd's algorithm [12] to compute the relaxation of 2D points. It takes as input a set of coordinates D and iteratively processes them to compute a Centroid Voronoi Tesselation [14] (CVT) configuration. This configuration implies that for a computed Voronoi diagram based on a set of 2D points, each generating point of a cell is also its centroid. This disposition results in evenly scattered points into the finite space. While its convergence has been proven in 1D spaces [13], the convergence of the algorithm in higher dimensional spaces, including 2D spaces, has yet to be proven. However, it has been proven [21] that in a bounded convex space, the algorithm always converges toward a CVT configuration. This condition is respected by the usage of convex hulls defined during the initialization step.

Lloyd's algorithm consists in repeating the following steps until stabilization: (1) The Voronoi diagram V is computed based on D. Each point $d \in D$ is associated with its corresponding Voronoi cell $v_d \in V$. (2) The centroid $C(v_d)$ of the cell v_d is computed. (3) Each point d is moved to their corresponding $C(v_d)$.
Stabilization is reached when the distance between each point d and their respective centroid $C(v_d)$ is smaller than a given ϵ. Smaller ϵ values give a more precise coordinate disposition and need more algorithm iterations.

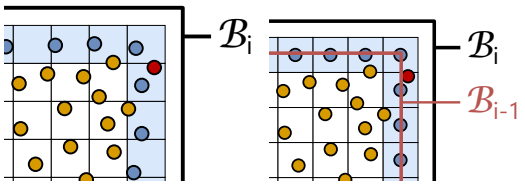
In our case initialized in the first step, Figure 4b shows stabilization reached after 223 iterations of Lloyd's algorithm on the sample dataset from the initialization step.

Since VRGrid only focuses on the position assignment of the points closest to the convex hull edges, we can impose a fixed number of iterations to Lloyd's algorithm to separate these candidates from the other points without waiting for complete



(a) Assignation to C_4 (b) Next iteration preparation

Fig. 5. Point assignation step (a) and preparation for next VRGrid iteration (b). Convex hull is changed to $\mathcal{B}_4$ and C_3 is the next set of cells to be assigned points to. Other P points are rescaled to a smaller scale to stay inside of $\mathcal{B}_4$.



(a) Assignation to C_{i-1} (b) Next iteration preparation

Fig. 6. Problematic configuration where a point (red circle) is not chosen to be assigned in C_{k-i-1} cells (blue circles) despite being close to $\mathcal{B}_{k-i}$. Because of that, when defining the new convex hull $\mathcal{B}_{k-i-1}$, the point is outside of it, and cannot be processed by the relaxation method as intended on the next VRGrid iteration. Rescaling unassigned data solves this issue.

stabilization. Doing so reduces the number of iterations and thus the overall computing time, but potentially lowers the final arrangement quality.

Point Assignation to Grid Cells. After the relaxation step (*i.e.*, once points are scattered enough into the current convex hull $\mathcal{B}_{k-i}$), we compute for each cell $c \in C_{k-i-1}$ its closest point $p \in P_i$. p is then assigned to c's position. In our case in Figure 4b, the assignment of closest points is represented by a segment toward their respective cell.

We then prepare the next iteration $i+1$ of VRGrid; convex hull $\mathcal{B}_{k-i-1}$ defines the area for the next application of the relaxation method and C_{k-i-2} the next grid cells to assign 2D points to. However, there is no guaranty that every point of P_{i+1} is inside $\mathcal{B}_{k-i-1}$ after C_{k-i-1} assignment. Indeed, the relaxation step is applied on the whole space delimited by $\mathcal{B}_{k-i-1}$, so any point of P_i could be close to the hull border and still not be assigned to a C_{k-i-1} cell. Since only $|C_{k-i-1}|$ points are assigned to grid cells, one or more points can be left out between the border of $\mathcal{B}_{k-i}$ and $\mathcal{B}_{k-i-1}$. Figure 6 presents an example where such a situation occurs.

To ensures this does not happen before the next relaxation algorithm application, we apply a rescale of P_{i+1} points to fit into the newest convex hull at the end of each VRGrid iteration. Figure 5 shows the assignment result following Lloyd's algorithm application and preparation for the next VRGrid's iteration.

End of VRGrid Iteration. Following the assignment of P_i points in C_{k-i-1} and the rescaling of remained unassigned points, the next iteration $i+1$ is applied (*i.e.*, application of the relaxation method in the $\mathcal{B}_{k-i-1}$ bounded space then assignment of P points into C_{k-i-2} cells). This iterative process is applied until all grid cells (except C_k cells) are assigned a point of the input dataset.

B. Complexity Analysis

This section details the computational complexity of VRGrid by inspecting each step of the algorithm. For better readability, we name n the size of the input dataset including eventual dummy points.

Initialization. VRGrid initialization step consists in building a set of convex hulls and rescaling input data of size n which can be summarized as the computational complexity of $O(n)$.

Relaxation Method. This part differs depending on the chosen relaxation method for VRGrid. In this paper, we make use of Lloyd's algorithm, which consists of three steps: computing the Voronoi diagram ($O(n.log(n))$), computing the centroid of each cell ($O(n)$) and moving coordinate to the corresponding cell's centroid ($O(n)$).

The computational complexity of each iteration of the algorithm is thus $O(n.log(n))$. The number of iterations i needed for Lloyd's algorithm depends on the density of the processed data (denser data needs more iterations than already evenly scattered data), and the stabilization factor ϵ (lower values and needs more iterations than higher values). As presented in the previous section, i can also be fixed to not wait for stabilization and thus lower the overall complexity of the relaxation method. For this relaxation method, the resulting computational complexity is $O(i.n.log(n))$.

Point assignment to cells. VRGrid's point assignment steps consist in looking in P for the closest points of each cell in C_i, assigning them and scaling the remaining unassigned P points to the next convex space. The nearest element of any coordinate in a 2D space can be efficiently computed using a quad-tree structure of $O(n.log(n))$ complexity. Assigning points and the scaling process are both $O(n)$. The point assignment step results in the computational complexity of $O(n.log(n))$.

VRGrid iterations. VRGrid re-applies the relaxation method until all $G \backslash C_k$ cells are used. The number of iterations is defined by the number of convex hulls k built during the initialization step. VRGrid thus needs to iterate $\lceil \sqrt{n}/2 \rceil + 1$ times. As a result, VRGrid's complexity is bound to $O\left(\sqrt{n}.i.n.log(n)\right) = O\left(i.n^{\frac{3}{2}}.log(n)\right)$.

IV. Experimental Protocol

We evaluate how VRGrid competes against state-of-the-art methods by looking at how well data topology is represented by the method and how different are the proposed arrangements from their input data. Five aspects are evaluated: pairwise distance preservation, neighborhood preservation, pairwise relative positioning, global positioning, and computation time.

In the evaluation, we focus on VRGrid, and the baselines Self Sorting Maps [9] and DGrid [10], [11]. We perform the evaluation in a context of large dataset processing ($\geq 1\,000$ elements). Self-Sorting Maps (SSM) is used and evaluated as a post-processing method, meaning its input data is **already projected, 2D data**.

VRGrid varies in its accuracy by changing the ϵ value and its number of fixed iterations i. Smaller ϵ values would result in more precise data representation but slower processing time while higher ϵ values would result in faster processing time but imprecise data positioning. Similarly, higher i values would result in better data representation at the expense of longer processing time, while lower i values would result in faster processing time for worse data positioning. In this evaluation, we test two configurations of VRGrid: VRGrid$_P$ is computing Lloyd Algorithm until stable configuration with an ϵ value fixed to 10^{-3}. VRGrid$_F$ is using a fixed number of iterations i which results in faster processing times at the cost of quality degradation. We fix i to 10 for Lloyd's algorithm to limit the relaxation mainly on the closest points of the convex hull border instead of the most denser point area.

A. Evaluation Metrics

The evaluation focuses on manifold understanding (reflected by pairwise distance preservation and neighborhood preservation) and drawing deformation minimization (depicted by relative and global positioning movements). We measure these aspects for each scenario using the following metrics:

Distance Preservation (DP) is evaluated using the Cross-Correlation metric to compare the distance between the grid arrangement and the 2D input data spaces. It is used by SSM as a loss function [9]. This comparison is done by computing pairwise distances in the input data and how they are correlated in the arrangement. Distances that are not correctly represented in the grid arrangement bring an empirical penalty to the evaluation. Higher DP values mean better distance preservation and thus better results.

Neighborhood Preservation (NP). As scatter plots are usually interpreted by how elements are positioned compared to the others, this metric rates how similar is each element's neighborhood in the grid arrangement to the input data.

Usually, neighborhood preservation is evaluated by computing the ratio of the k-nearest neighbors for each element between the input dataset and the projection. As this evaluation is focused on grid arrangements, we consider the neighborhood $knnG(k, i)$ of the element i to be composed of the elements around i within a square area of k cells size, including diagonals. This also means that elements close to the border of the grid have a smaller neighborhood than others. This neighborhood is then compared with the input data using the same number of neighbors. The higher this value is, the better the neighborhood is preserved.

Relative Positioning Preservation (RPP) is computed using the Orthogonal Ordering metric primarily designed to evaluate shape preservation of overlap solvers in graph drawing [22]. We use this metric to evaluate how well the position of each element relative to others is preserved before and after computing the grid arrangement. Lower RPP values mean better preservation of elements relative to others and thus better results, while higher values mean higher deformation and thus worst results.

Global Positioning Preservation (GPP) is being computed using the Node Movements metric. It evaluates how coordinates moved from the original projection to the resulting arrangement [22]. While coordinate movements are unavoidable if the shape of the projection is altered, we assume the faithful grid arrangement of 2D data involves minimal movements. Lower GPP values mean smaller alterations from the original positions and thus better results, while higher values mean worst results.

TABLE I
EVALUATED DATASET SUMMARY

Name	Size × Dimension	Source
Swiss	1 024 × 3	[23]
Broken swiss	1 024 × 3	[23]
Helix	1 024 × 3	[23]
Twinpeaks	1 024 × 3	[23]
MNIST	10 000 × 784	[24]
Fashion	10 000 × 784	[25]

B. Datasets and Visualization Methods

Datasets. In the evaluation, we use datasets varying in sizes, dimensions, and natures, as referenced in Table I. Datasets "Swissroll", "Broken swissroll", "Helix" and "Twinpeaks", generated with the *Matlab Toolbox for Dimensionality Reduction* [23], have very specific shapes in a 3D space and are often tested in dimension reduction research field, including SSM [9]. Their shape reveals the strategies of the projection methods and their limitation. To reflect actual real-life scenarios, we also proceed with our evaluation on real-life datasets MNIST [24] and Fashion-MNIST [25] which are both commonly used in the machine learning field. Using t-SNE [26], [27], UMAP [8], MDS [4], [27] and IsoMap [19], [27], we then project the datasets in a 2D space to generate our evaluation datasets. Every combination result of the datasets presented in Table I and are called **real datasets** in our evaluation.

We also proceed with our evaluation on 800 random 2D datasets varying in sizes (64, 100, 1 000 and 10 000 points) and scattering distribution methods (evenly scattered using Permuted Congruential Generator [28] and dense using normal distribution). These datasets are used to evaluate the stability of the grid arrangement methods.

Data Visualization. In this evaluation, we focus on how well evaluated methods can represent their 2D input data into a grid. Dimension reduction methods used to generate testing datasets are renowned for their efficiency, and it is common to colorize projected data using class information. This colorization usually reveals data clusters and outliers. However, well-clustered points can hardly be visually compared against each other since they have the same color. To visually evaluate how grid arrangement result differs from its input data, we colorize arranged points according to their initial 2D positions. Doing so shows initially close points with a similar hue while initially distant points will have a strongly different color. We can thus compare how points moved compared to their neighbors after the arrangement even if they were inside the same cluster.

Self Sorting Map and DGrid Implementations. At the time of the redaction of this paper, the SSM algorithm implementation of its authors is no more available [29]. The

TABLE II
COMPUTED p-VALUES OF THE WILCOXON SIGNED RANK TEST PERFORMED ON VRGRID$_P$ AND VRGRID$_F$ AGAINST SSM AND DGRID

Method	Metric	SSM	DGrid
VRGrid$_P$	DP	3.52×10^{-56}	6.51×10^{-67}
	NP	1.03×10^{-83}	7.20×10^{-34}
	RPP	6.14×10^{-125}	1.37×10^{-27}
	GPP	2.55×10^{-125}	4.20×10^{-43}
VRGrid$_F$	DP	2.00×10^{-4}	1.66×10^{-28}
	NP	7.71×10^{-114}	1.93×10^{-6}
	RPP	1.70×10^{-89}	6.17×10^{-19}
	GPP	2.01×10^{-104}	3.92×10^{-7}

SSM algorithm used in this study is implemented in Rust following the directives of the authors' paper [9]. According to our evaluation of the Cross-Correlation Score, which is also used in the authors' paper, the quality of our produced results is very close to the authors' algorithm performances.

Both Orthogonal Ordering and Node Movement Minimization metrics are comparing the positioning of coordinates between the original projection and grid arrangement, which preservation is not guaranteed by the SSM method by design. To evaluate SSM with RPP and GPP, we automatically correct SSM results by applying all possible orthogonal transformations to the arrangement: flip of grid cells on X-axis and/or Y-axis and retaining the configuration which has the best GPP value.

DGrid has been implemented following the algorithm presentation given by the authors [11] in Python. Obtained results in both computation times and neighborhood preservation correlate with the performances presented in the authors' paper [10].

V. RESULTS AND DISCUSSION

We compare methods across the five metrics presented in Section IV-A and present several results computed on real data, colorized accordingly to the original dataset labeling and their 2D position in the input dataset in Figure 1. To evaluate whenever a method is significantly better than the other on a given metric, we make use of the Wilcoxon signed rank statistical test [30] computed on the random datasets. If the computed p-value is lower than 5×10^{-2}, then the results are significantly different. Wilcoxon signed rank test's p-values between VRGrid configurations and both SSM and DGrid are presented in Table II and are all lower than 5×10^{-2}. All differences between VRGrid$_P$, VRGrid$_F$, SSM and DGrid evaluations are thus statistically significant.

Visual Inspection. Input data are identifiable in VRGrid$_P$ and VRGrid$_F$ results, the initial positions of clusters and their composition are preserved in the grid arrangements. However, VRGrid$_F$ has distortions along both diagonals especially noticeable using position-based colorization. VRGrid$_P$ has no such visible distortions and has a better-looking separation between the different element classes. In SSM arrangements, the position of the different item groups is slightly different compared to the input data, while still being recognizable (*e.g.*, UMAP projection of MNIST and t-SNE projection of Fashion dataset are slightly rotated clockwise by SSM compared to the input data). Aside from that, SSM arrangements look quite like VRGrid$_P$'s with sharper borders between each data class. However, when using position-based colorization, we can notice a few elements far from their original neighbors, notably in the UMAP/Swissroll scenario. In DGrid arrangements, the partitioning strategy is very noticeable on MNIST and Fashion results in the form of small square-shaped areas. In the Swissroll/UMAP arrangement, the placement of the red classes in the lower-left part of the grid does not seem intuitive. Position-based colorization reveals partitioning artifacts even more which can disturb data visualization interpretation.

Distance Preservation (DP) is computed using Cross-Correlation and results are depicted in Figure 7a, with higher values meaning better pairwise distance preservation. On real datasets, distances are well preserved across all methods. VRGrid$_P$ produces the best results overall with values about 2.57% better than SSM, 2.61% better than DGrid and 5.41% better than VRGrid$_F$. On randomly generated datasets, VRGrid$_P$ is also better with results about 1.77% better than SSM, 6.73% better than DGrid and 2.63% better than VRGrid$_F$. It can be concluded that all four methods manage to preserve distances when arranging data into a grid on real datasets, but VRGrid$_P$ performs better.

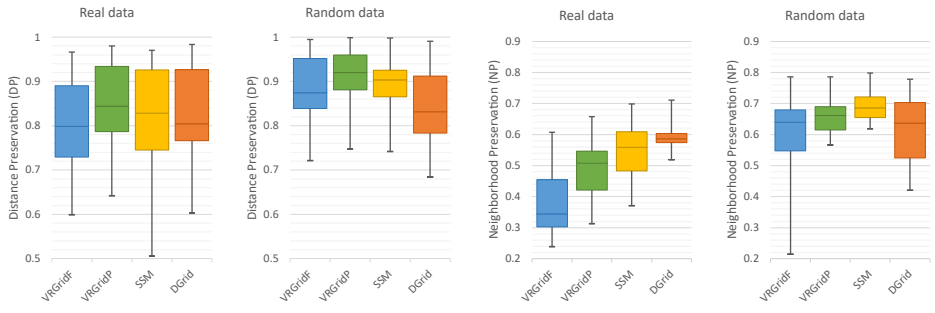
Neighborhood Preservation (NP) results are depicted in Figure 7b, with higher values meaning better neighborhood preservation across elements. On real datasets, DGrid produces the best results which are overall 8.94% better than SSM, 19.44% better than VRGrid$_P$ and 37.52% better than VRGrid$_F$. On random datasets, SSM produces the best arrangement with values about 4.76% better than VRGrid$_P$, 12.16% better than DGrid and 16.19% better than VRGrid$_F$.

SSM is better than VRGrid$_P$ at preserving neighborhoods on both real and randomly generated data. DGrid is better than VRGrid$_P$ on real data but not on randomly generated data. VRGrid$_F$ has the worst results on both real and random datasets.

Differences in quality between methods can be visually observed in proposed arrangements in Figures 1 where SSM separates different data classes with sharp separations between groups while VRGrid$_P$ and VRGrid$_F$ slightly mix those separations. As for DGrid, despite its locally focused arrangements, the method preserves neighborhood very well on real data but not as well on random data, mainly on random datasets of size $10\,000$ regardless of the distribution method. This behavior requires additional investigation as it seems that the partitioning technique of DGrid handles badly grids of sizes different than a power of 2, yet the method still performs well with MNIST and Fashion-MNIST which are the same size.

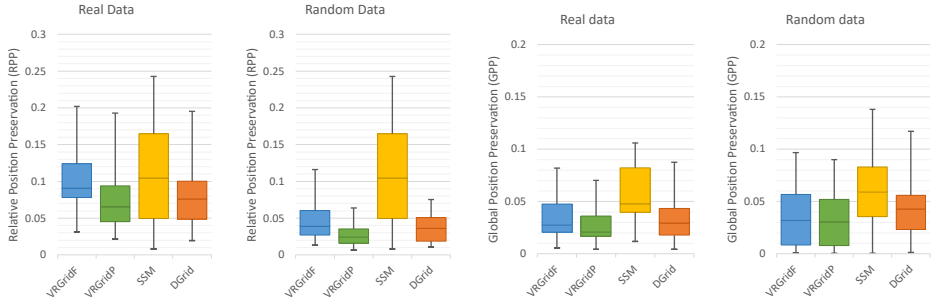
Relative Position Preservation (RPP)

values are presented in Figure 7c, with lower values meaning lower local permutations in the grid arrangement resulting in better representation of input data. On real data, relative positioning in VRGrid$_P$ arrangements are conserved about



(a) Distance Preservation measure.

(b) Neighborhood Preservation measure.



(c) Relative Position Preservation measures.

(d) Global Position Preservation measure.

Fig. 7. Evaluation results for real and random datasets. For Distance and Neighborhood Preservation measures, higher values are better. For Relative and Global Preservation measures, lower values are better.

44.22% better than SSM, 6.53% better than DGrid and 29.04% better than VRGrid$_F$. On randomly generated data, VRGrid$_P$ is about 62.04% better than SSM, 15.01% better than DGrid and 34.04% better than VRGrid$_F$. SSM's poor performances can be visually observed on produced arrangements in Figure 1 where arrangements are slightly rotated and distorted compared to the input data, when VRGrid$_P$, VRGrid$_F$ and DGrid manage to preserve the overall positioning of the elements in their arrangements.

By design, SSM does not preserve the input data positions, which can explain its poor performance in that aspect of the evaluation. DGrid has retention information but is outperformed by VRGrid$_P$.

Global Position Preservation (GPP) values are presented in Figure 7d, with lower values meaning less overall deformation of the original data projection. On real datasets, VRGrid$_P$ produces arrangement with 46.59% less movements than SSM, 14.83% less than DGrid and 21.60% less than VRGrid$_F$. On random datasets, VRGrid$_P$ arrangements are also 46.47% better than SSM, 23.55% better than DGrid and 17.66% better than VRGrid$_F$. Thus, VRGrid$_P$ produces an arrangement with fewer distortions than other methods.

Observed GPP values also show that the fragmented aspect of DGrid arrangements or the distortions in VRGrid$_F$ arrangements is as highly penalized as SSM.

Computation Time are presented in Table III, with lower values meaning faster processing. When used as a post-processing method, SSM yields faster computation times

TABLE III
AVERAGE COMPUTATION TIME[1] ON TESTED DATASETS[2]

Dataset type	Dataset size	VRGrid$_F$	VRGrid$_P$	SSM	DGrid
Real	1024	1.65	66.94	0.93	<0.01
	10 000	46.17	5594.37	16.13	0.01
Random (dense)	64	0.04	0.46	0.02	<0.01
	100	0.07	1.01	0.03	<0.01
	1000	1.65	68.53	0.52	<0.01
	10 000	46.94	5628.95	15.92	0.01
Random (scattered)	64	0.05	0.40	0.03	<0.01
	100	0.07	0.98	0.04	<0.01
	1000	1.64	63.10	0.55	<0.01
	10 000	46.57	4999.92	9.46	0.01

[1] in seconds, lower values are better.
[2] organized by their nature and sizes, see Section IV-B.

than VRGrid. DGrid surpasses all other methods as it takes less than 0.02 second to process the largest datasets in our evaluation. VRGrid$_P$, despite its overall good quality results except in neighborhood preservation, is about 270 times slower than SSM and cannot compete to DGrid, making it currently hardly practical for large datasets such as MNIST by taking around 90 minutes to process against 15 seconds for SSM and 0.015 second for DGrid. By not limiting the number of iterations, VRGrid$_P$ can reach up to ten times the number of processed elements, which is hardly comparable to VRGrid$_F$at larger scales. Furthermore, density of data seems to have a lesser impact on VRGrid$_P$ total computation

time with a 11% difference between dense and scattered data. VRGrid$_F$ computation times are between up to 100 times faster than VRGrid$_P$, but are still slower than SSM.

Discussion. The evaluation shows that VRGrid$_P$ brings better results than SSM, DGrid and VRGrid$_F$ on distance preservation (DP) and minimizes deformation of the initial data (RPP and GPP). However, neighborhood preservation (NP) is a weak point of VRGrid$_P$, as it is outclassed by both SSM and DGrid despite its longer processing time.

SSM is significantly weaker at globally staying faithful to the original data projection shape. Indeed, even using minimal deformation configuration of SSM's results explained in Section IV-B, both positioning metrics show that SSM arrangements are more distorted than VRGrid$_P$, sometimes producing sources of visual misinterpretation as shown in Figure 1 between the two colorization methods of the UMAP/Swissroll scenario (first columns, second row). While DGrid evaluation values are closer yet lower than VRGrid$_P$ in retaining input data shape, the visualization method proposed in Section IV-B shows that DGrid's arrangements are locally very fragmented compared to other methods. However, this distortion is not very apparent in our evaluation results, notably by the RPP and GPP metrics, whereas SSM's outliers seem to be detected and penalized accordingly, as seen in the UMAP/Swissroll scenario. A dedicated novel metric that detects this issue should be designed in future works.

With those observations, we can affirm that all methods are useful in different usage scenarios. Scenarios only requiring compact projection of data (*i.e.*, ignoring input data's shape) can already make good use of SSM as a post-processing method, since neighborhood and distances are well preserved with reasonable processing time. However, VRGrid and DGrid are useful for producing compact projections as an additional tool to the regular data projection, such as data overlap solving tasks and/or numerous dataset projection navigation in a dynamic environment. In that aspect, we showed that, given a larger processing time, VRGrid brings significantly better results than DGrid, and does process its arrangement globally on its input data compared to DGrid's locally focused strategy. Dynamic or multi-view visualization tools would better benefit from VRGrid arrangements than other tested methods' as VRGrid produces overall fewer distortions.

Finally, our evaluation shows that VRGrid can hardly compete against SSM and DGrid for the fastest processing methods with its current implementation, the number of iterations needed for VRGrid$_P$ is of the same order as the dataset size n if not higher, which implies a computational complexity of $O\left(n^{\frac{5}{2}}.log(n)\right)$. The cut in computation time of VRGrid$_F$ by imposing a fixed number of iterations is hindering the overall quality of the arrangement visually and is apparent in the evaluation metrics but reduces the computation time by up to 100 times. But this tradeoff in quality is still insufficient to surpass SSM on larger datasets.

As it stands now, VRGrid$_P$ would benefit from improvement on Lloyd's algorithm implementation, as cutting down iteration count results in a large decrease in overall arrangement quality. Lloyd's algorithm had several ways to be improved with algorithmic techniques such as over-relaxation [31] and hardware optimization [32], [33]. Those improvements led the computation of CVT to at least 10 times faster than the classic Lloyd's algorithm running on CPU. This improvement potential could be a way to mitigate the high computational complexity of VRGrid$_P$ and make it usable for bigger dataset processing.

VI. Conclusion

When visualizing projected data, overlap may occur and hinder the data analysis. Clustering methods such as t-SNE are popular high-dimension reduction techniques emphasizing even more visual occlusion in their results. Pixel-oriented visualization techniques, such as grid arrangement methods, are an efficient way to eliminate this visual occlusion and improve the perceptual scalability of dimension reduction methods.

We presented VRGrid, a post-processing technique using relaxation methods to provide a grid arrangement for 2D data. VRGrid consumes the position of input data and incrementally transforms them into a grid disposition while limiting the deformation of the initial data shape. We evaluated this method against state-of-the-art methods Self-Sorting Maps and DGrid on several aspects focused on data understanding and minimizing distortions.

In this evaluation, we showed that VRGrid using Lloyd's algorithm surpasses SSM and DGrid with better distance preservation and minimal deformation but falls short in neighborhood preservation and more specifically in computing time. We showed that limiting the number of Lloyd's algorithm iterations significantly reduces processing time but at the cost of a grid arrangement of lower quality across all four metrics.

These two observations showed that the usage of a relaxation method by VRGrid to converge toward a CVT configuration can be used to produce a grid arrangement for 2D data. To prevent deformations seen with VRGrid$_F$, the relaxation method must process enough iterations before VRGrid's iterative border-building step. This requirement may result in larger computation time, as seen with VRGrid$_P$, which is not desirable when working with large datasets. Lloyd's algorithm implementation can already be improved in several ways thanks to previous works in the domain. Other relaxation techniques can also be used with VRGrid, Broyden-Fletcher-Goldfarb-Shanno method (or BFGS) and its variants [34] are optimization methods that can be used to compute CVT configurations with fewer iterations than Lloyd's algorithm. Finally, RPP and GPP metrics were efficient to detect misarranged elements by SSM in the resulting grid but have not penalized VRGrid$_F$ and DGrid as much for their visually different results from VRGrid$_P$. Further work on the evaluation of grid arrangement quality and practicability would be useful as the three methods may be interpreted differently by the end-user if given as is.

Acknowledgment

This work has been carried out with financial support from the French State, managed by the French National Research

Agency (ANR) in the frame of the Involvd project (ANR-20-CE23-0023-04). Experiments presented in this paper were carried out using the Labo's in the Sky with Data(LSD), the LaBRI data platform partially funded by Region Nouvelle Aquitaine, and using PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/).

REFERENCES

[1] M. Friendly, "A brief history of data visualization," in *Handbook of data visualization*. Springer, 2008, pp. 15–56.

[2] S. Liu, W. Cui, Y. Wu, and M. Liu, "A survey on information visualization: recent advances and challenges," *The Visual Computer*, vol. 30, no. 12, pp. 1373–1393, 2014.

[3] J. Bertin, "Sémiologie graphique: Les diagrammes-les réseaux-les cartes," Gauthier-VillarsMouton & Cie, Tech. Rep., 1973.

[4] D. Engel, L. Hüttenberger, and B. Hamann, "A survey of dimension reduction methods for high-dimensional data analysis and visualization," in *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering-Proceedings of IRTG 1131 Workshop 2011*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[5] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci, "Visualizing high-dimensional data: Advances in the past decade," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 3, pp. 1249–1268, 2016.

[6] D. L. Donoho *et al.*, "High-dimensional data analysis: The curses and blessings of dimensionality," *AMS math challenges lecture*, vol. 1, no. 2000, p. 32, 2000.

[7] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[8] L. McInnes, J. Healy, N. Saul, and L. Grossberger, "Umap: Uniform manifold approximation and projection," *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[9] G. Strong and M. Gong, "Self-sorting map: An efficient algorithm for presenting multimedia data in structured layouts," *IEEE Transactions on Multimedia*, vol. 16, no. 4, pp. 1045–1058, 2014.

[10] G. M. Hilasaca and F. V. Paulovich, "A visual approach for user-guided feature fusion," in *Anais Estendidos da XXXII Conference on Graphics, Patterns and Images*. SBC, 2019, pp. 133–139.

[11] G. M. H. Mamani, "A visual approach for user-guided feature fusion," Ph.D. dissertation, Universidade de São Paulo.

[12] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[13] Q. Du, M. Emelianenko, and L. Ju, "Convergence of the lloyd algorithm for computing centroidal voronoi tessellations," *SIAM journal on numerical analysis*, vol. 44, no. 1, pp. 102–119, 2006.

[14] Q. Du, V. Faber, and M. Gunzburger, "Centroidal voronoi tessellations: Applications and algorithms," *SIAM review*, vol. 41, no. 4, pp. 637–676, 1999.

[15] D. Hilbert, "Über die stetige abbildung einer linie auf ein flächenstück," in *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes*. Springer, 1935, pp. 1–2.

[16] D. A. Keim, "Designing pixel-oriented visualization techniques: Theory and applications," *IEEE Transactions on visualization and computer graphics*, vol. 6, no. 1, pp. 59–78, 2000.

[17] D. Auber, C. Huet, A. Lambert, B. Renoust, A. Sallaberry, and A. Saulnier, "Gospermap: Using a gosper curve for laying out hierarchical data," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 11, pp. 1820–1832, 2013.

[18] O. Fried, S. DiVerdi, M. Halber, E. Sizikova, and A. Finkelstein, "Isomatch: Creating informative grid layouts," in *Computer graphics forum*, vol. 34, no. 2. Wiley Online Library, 2015, pp. 155–166.

[19] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[20] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[21] M. Emelianenko, L. Ju, and A. Rand, "Nondegeneracy and weak global convergence of the lloyd algorithm in r^d," *SIAM Journal on Numerical Analysis*, vol. 46, no. 3, pp. 1423–1441, 2008.

[22] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry, "Node overlap removal algorithms: an extended comparative study," *Journal of Graph Algorithms and Applications*, 2020.

[23] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, no. 66-71, p. 13, 2009.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[26] L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] M. E. O'Neill, "Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation," *ACM Transactions on Mathematical Software*, 2014.

[29] M. Gong, "Organize data into structured layout," http://www.cs.mun.ca/ gong/research/DataOrganization.html, 2019.

[30] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[31] X. Xiao, *Over-relaxation Lloyd method for computing centroidal Voronoi tessellations*. University of South Carolina, 2010.

[32] G. Rong, Y. Liu, W. Wang, X. Yin, D. Gu, and X. Guo, "Gpu-assisted computation of centroidal voronoi tessellation," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 3, pp. 345–356, 2010.

[33] J. Zheng and T.-S. Tan, "Computing centroidal voronoi tessellation using the gpu," in *Symposium on Interactive 3D Graphics and Games*, 2020, pp. 1–9.

[34] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang, "On centroidal voronoi tessellation—energy smoothness and fast computation," *ACM Transactions on Graphics (ToG)*, vol. 28, no. 4, pp. 1–17, 2009.