Guaranteed Visibility in Scatterplots with Tolerance

Loann Giovannangeli, Frédéric Lalanne, Romain Giot, Romain Bourqui

Guaranteed Visibility in Scatterplots with Tolerance

Loann Giovannangeli, Frederic Lalanne, Romain Giot and Romain Bourqui

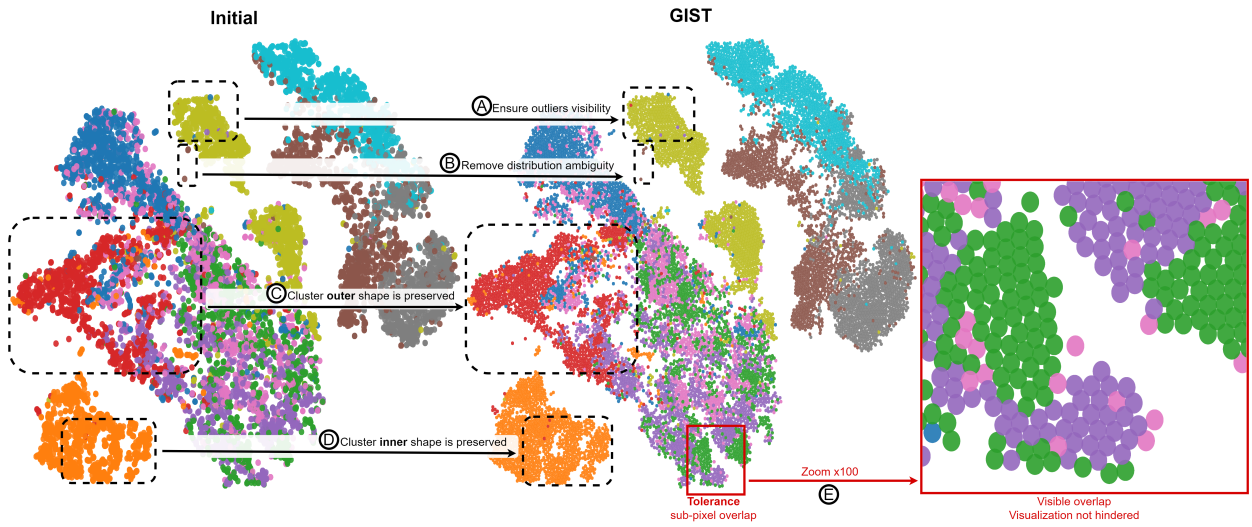

Fig. 1: Result of our method, GIST, on a dataset of 10000 points with a resolution of 2000×2000. Our algorithm guarantees that all nodes are visible in the given resolution ⒶⒷ and maximizes both the node sizes and the preservation of the initial layout ⒸⒹ. GIST convergence is greatly eased by the introduction of a *tolerance* factor to overlaps Ⓔ. The tolerance is minimized for the given resolution and is at most 1 pixel large. Even zoomed in, the tolerance remains harmless.

Abstract—In 2D visualizations, visibility of every datum's representation is crucial to ease the completion of visual tasks. Such a guarantee is barely respected in complex visualizations, mainly because of overdraws between datum representations that hide parts of the information (e.g., outliers). The literature proposes various Layout Adjustment algorithms to improve the readability of visualizations that suffer from this issue. Manipulating the data in high-dimensional, geometric or visual space; they rely on different strategies with their own strengths and weaknesses. Moreover, most of these algorithms are computationally expensive as they search for an exact solution in the geometric space and do not scale well to large datasets.
This article proposes GIST, a layout adjustment algorithm that aims at optimizing three criteria: (i) node visibility guarantee (at least 1 pixel), (ii) node size maximization, and (iii) the original layout preservation. This is achieved by combining a search for the maximum node size that enables to draw all the data points without overlaps, with a limited budget of movements (*i.e.*, limiting the distortions of the original layout). The method's basis relies on the idea that it is not necessary for two data representations to be strictly not overlapping in order to guarantee their visibility in visual space. Our algorithm therefore uses a tolerance in the geometric space to determine the overlaps between pairs of data. The tolerance is optimized such that the approximation computed in the geometric space can lead to visualization without noticeable overdraw after the data rendering rasterization. In addition, such an approximation helps to ease the algorithm's convergence as it reduces the number of constraints to resolve, enabling it to handle large datasets. We demonstrate the effectiveness of our approach by comparing its results to those of state-of-the-art methods on several large datasets.

Index Terms—Guaranteed visibility, Layout adjustment, Overlap removal, Scatterplots

1 Introduction

Multidimensional data is frequently used in industrial scenarios (*e.g.*, web analytics, biology, data science). The amount of data and their number of dimensions becomes larger and larger, especially since the rise of modern AI techniques that process datasets of thousands or even millions of data samples. For instance, understanding the predictions made by AI models systematically implies to study their input data to understand their semantic. Hence, visualizing and exploring projections of the models inputs or predictions, looking for clusters and outliers (*e.g.*, misclassifications) [18, 19, 28, 37], becomes common. It therefore becomes mandatory to design efficient visualizations to support their exploration. One of the most admitted visualization techniques is the scatterplot which represents data points as shapes in a 2D plan (or 3D volume). The coordinates of these data points either represent their values in a 2D sub-space of the original high-dimensional space; or are computed using a dimensionality reduction technique (*e.g.*, t-SNE [35], UMAP [25]). However, scatterplots suffer from an occlusion issue that straightforward methods such as opacity modulation cannot solve as soon as the amount of data increases. The use of a 3D representation with the ability to move the camera[1] does not solve the issue when the data is too complex (*i.e.*, many points with high density areas). In addition, 3D visualizations basically cannot resolve the occlusion issue on their own as the medium on which they are consulted, *i.e.*, monitors, necessarily flatten the rendered image. This hinders the understanding of the visualization as some portion of the information is hidden to the user. In particular, perception of local densities or

• *All authors are with the Univ. Bordeaux, CNRS, Bordeaux INP, INRIA, LaBRI, UMR 5800, F-33400 Talence, France.*
E-mail: {firstname}.{lastname}@u-bordeaux.fr

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

[1] https://projector.tensorflow.org/

outlier identification in dense regions quickly becomes troublesome if not impossible, whereas cluster and outlier identification are among the most common tasks on complex data.

Many methods have been proposed in the literature to overcome the overdrawing issue [6–10, 14–16, 24, 29, 30, 33]. These methods belong to two categories depending on the space containing the data they manipulate. In *Geometric Space* (GS) methods, the manipulated data consists in a set of positions in the 2D plan associated to their size, $\{\{x,y,r\},x,y,r \in \mathbb{R}\}$. On the other hand, *Visual Space* (VS) methods can also use the screen resolution, data point color or any other rendering option to enhance the visualization of all data points. This means that GS is included in VS. As one of the key goals of our work is to enhance the visibility of each and every data point in an image, this paper considers methods operating in VS. They do not have any knowledge about the high-dimensional space that led into a scatterplot, as opposed to *Data space* approaches (*e.g.*, sampling [1, 5, 38]). The GS methods results are projected in a VS by discretizing the space with a fixed resolution, which can then be interpreted as a rasterized image.

GS methods [7–10, 14, 16, 29, 30, 33] produce exact solutions to the overlap removal problem. While some of these methods strongly rely on the *scan line* algorithm to process the data points sequentially (*e.g.*, [14, 16, 29, 33]), others model the overlap removal problem as a stress optimization (*e.g.*, [9, 10, 30]). However, none of them have sub-quadratic complexity and they hardly scale well to large datasets. This paper relaxes the strictness of the overlap removal task to focus on the visibility of all data points. This approximation allows a better scalability by reducing the computational cost.

In addition to data point positions, VS methods [6, 15, 24] also have access to rendering parameters such as screen resolution, opacity or data point screen sizes. In DGrid [15] and HaGrid [6] for instance, a grid (screen) resolution is used to determine candidate positions such that data points do not overlap. In ScatterplotUnfold [24], rendered data point sizes are non-uniformly modified to remove overlaps and enhance the visibility of outliers in sparse regions of the drawing. These methods may have different objectives. For instance, DGrid generates grid-like visualizations particularly compact in dense regions of data points whereas the primary objective of ScatterplotUnfold is to preserve the perception of the data points density distribution in the visualization.

Any point set visualization can be processed by these approaches (*e.g.*, scatterplot, symbol map). However, we focus on data projected by dimension reduction algorithms because most approaches distort the original data layout in a way that would be very harmful to non projected 2D data. This is particularly true when considering data where dimensions have a semantic meaning (*e.g.*, X-axis encodes chronological years), methods that do not fully preserve the data orthogonal order will produce a misleading visualization as inversions on the X-axis can lead to severe misinterpretations. Such inversions are less harmful with multidimensional data projections as the axes may not have a meaning, the main information being the distance distribution of the elements.

The method presented in this paper, GIST (Guaranteed Visibility in Scatterplots with Tolerance), belongs to the VS approach. The algorithm focuses on data points' visibility in a resolution taken as input. More precisely, GIST optimizes three criteria: (i) nodes must be visible with at least 1 pixel in a given resolution R, (ii) nodes must be as wide as possible, and (iii) the initial layout must be preserved. The interactive demonstration of GIST, available online in [11], illustrates on multiple datasets the algorithm's capability to adjust layouts to maximize the nodes visibility on screen. As shown in the provided examples, the overlap tolerance does not hinder the visualization.

The main contributions of the article can be summarized as follows:

- We propose GIST, an approximation of overlap removal algorithm that produces a layout where data points can overlap up to a certain *tolerance*. It therefore acts as a layout adjustment method that proposes a unique trade-off between data points visibility and initial layout preservation for a target screen resolution. When data point visibility cannot be guaranteed at the target resolution, GIST automatically finds a higher resolution to solve the task.
- We present the results of a quantitative comparison of 6 state-of-the-art methods on a set of 48 scatterplots whose sizes vary between 1000 and 150000 data points. These results are compared to GIST results and emphasize the performance of the proposed method.

The remainder of the article is organized as follows. Section 2 presents related works on GS and VS Overlap Removal methods. Section 3 describes how GIST is built, while Section 4 presents its quantitative evaluation. Section 5 discusses some visual examples of the Layout Adjustment methods, as well as some of GIST's behaviors and limitations observed during the evaluation. Finally, Section 6 concludes the paper and proposes future works leads.

Notations: In the next, we use the word *node* to refer to the *data points* as it is commonly admitted in the Layout Adjustment literature. Let $X \in \mathbb{R}^{N\times 2}$ be a layout of N nodes, where X_i is the position of the node i in 2D. The radius of the circle representing node i is noted r_i. An input layout is denoted X^0, while an adjusted solution is referred to as X'. The euclidean distance between nodes i and j is noted $d(X_i,X_j)$.

2 Related Works

This section presents state-of-the-art algorithms for both Geometric Space (GS) and Visual Space (VS) approaches to overdraws removal. As we define them, the GS is included in the VS. The distinction between them is VS awareness of the rendering environment.

2.1 Geometric Space Algorithms (Nodes Dispersion)

A GS overdraw removal algorithm takes a set of positions and sizes in some defined 2D space, and aims at strictly removing all the intersections between the nodes. Such algorithms only take as input the coordinates and sizes of the nodes. The main strategy to solve the task is *Nodes Dispersion* where the initial layout is considered as a reference distribution in the GS, and in which some constraints (*i.e.*, overlaps) must be relaxed. This is achieved by moving the nodes, with the additional goal to preserve the reference distribution in the GS.

A review of the Nodes Dispersion literature was presented in Chen *et al*'s survey [3, 4]. PFS [29], PFS' [14], FTA [16], RWordle-L [33] are orthogonal iterative algorithms that rely on *scan-line* [7] to identify overlaps in a layout and process them. VPSC [7, 8] and Diamond [26] are two constraint-based algorithms that explicitly model the overlaps (or non-overlaps) as a set of constraints to relax and optimize. Finally, PRISM [9], GTree [30] and FORBID [10] model the distances to preserve (*i.e.*, between non-overlapped nodes) and those to satisfy (*i.e.*, between overlapped nodes) as a *stress* function to optimize. *Stress* is usually denoted σ and defined as:

$$\sigma(X) = \sum_{i,j\in X} W_{ij}\left(d\left(X_i,X_j\right) - \delta_{ij}\right)^2 \tag{1}$$

where δ_{ij} is an *ideal distance* defined between every pair of nodes according to some criteria, usually in the high-dimensional space of the original data (*e.g.*, MultiDimensional Scaling [22], Graph Layout [2, 20, 32]). W_{ij} is a weight factor commonly set to δ_{ij}^{-2} such that it is more important to preserve the shortest distances than the longest ones. The adaptation to Overlap Removal by Nodes Dispersion is achieved by defining the ideal distances such that some distance is added between overlapped nodes, while the distance between non-overlapped nodes must be respected. It is then the role of a stress-optimization algorithm to relax the model's forces.

FORBID [10] is the closest method to GIST as it was used as a backbone to build our algorithm. The algorithm's originality is to interlace two main components: a binary search for an optimal scale ratio, and Nodes Dispersion modeled as stress and optimized by Stochastic Gradient Descent [40]. The combination of these blocks enable it to search a compromise between the preservation of the initial layout, and the overlap-free layout's compactness.

The main problem of these techniques is their computation's expensiveness. Their complexity is quadratic on the number of nodes and the requirement to strictly remove overlaps in the GS makes them unable to efficiently scale to large datasets. Nevertheless, they remain state-of-the-art in the Overdraw Removal field as, by design, they guarantee to

provide an overlap-free layout. Yet, it is common for such overlap-free layouts to have dispersed the nodes so much that representing them in a reasonable resolution makes them less than 1 pixel wide, and therefore impossible to perceive.

In this article, we propose GIST to alleviate these flaws by leveraging one of these efficient Nodes Dispersion algorithms and making it scale to larger datasets while constraining the produced layout in a range where the nodes remain visible once projected into a reasonable resolution for large datasets (*e.g.*, 2000px wide).

2.2 Visual Space Algorithms

We call Visual Space (VS) algorithms the approaches in which data rendering parameters (*e.g.*, screen resolution, visual encoding such as color and opacity) are available and even optimizable. In addition, while GS algorithms aim at strictly satisfying some criteria, we argue that VS methods can leverage approximations to make sure that the produced visualization conveys the information encoded within the original data. These approximations lead to distortions whose amplitude should be quantified and minimized, but also enable faster processing.

ScatterplotUnfold [24] is an illustration of VS algorithms. Its goal is to make sure that the produced visualization preserves the distribution of node *densities* of the input layout while also trying to highlight outliers. To achieve its goal, the algorithm moves nodes using a polar packing algorithm and modifies their diameter to emphasize the node *densities*. The produced visualization is made of nodes with various diameters, some of them not even being 1 pixel wide. Overall, it does efficiently preserve the distribution of densities of the original layout while removing overdraws.

We also include grid-based algorithms in the VS category. By design, such algorithms optimize node placements according to a grid size that is directly or indirectly related to the produced screen resolution of the visualization. DGrid [15] and HaGrid [6] are two examples of grid-based overdraw removal algorithms. DGrid [15] recursively splits the VS into subspaces in which nodes cannot overlap. In HaGrid [6], the node coordinates are assigned using a space-filling curve whose recursion depth is set to produce more or less compact layouts. While these methods have a $\mathcal{O}(Nlog(N))$ time complexity, they produce significant deformation of the initial layout, in particular in dense regions.

Finally, we can include compact visualization algorithms in this category as they also try to limit their VS usage (*i.e.*, resolution) by producing visualizations as compact as possible. By design, they also remove overdraws by assigning every node to an exclusive position. These algorithms contrast with the grid-based algorithms mentioned above as they *do* try to minimize the grid size in which they produce the visualization. This is not necessarily the case with grid-based approaches for whom the target resolution can be modified in some way. SSM [34] is a compact visualization algorithm that randomly arranges nodes into a grid and swaps them until the layout satisfies some dissimilarity metric. VRGrid [13] is another example of compact arrangement method that computes Voronoi Tesselations to split the VS and assign a cell and position to every node.

We consider that GIST is an approximate Node Dispersion technique in VS since it manipulates rendering parameters with the image resolution, node diameters and tolerance in pixels. Nevertheless, it also leverages a Node Dispersion technique in the GS to optimize node movements and produce visualizations where nodes remain visible.

3 GIST Algorithm

This section describes GIST (Guaranteed vIsibility in Scatterplots with Tolerance). The algorithm is built on two interlaced components: node movements, and the search for the optimal node representation radius. Its goal is to produce a layout in which (i) nodes are visible with at least 1 pixel in a given resolution R, (ii) nodes are the largest possible, and (iii) the initial layout is preserved. Finding a solution that proposes a compromise between these criteria is what we later refer to as *solving the task*. More specifically, GIST solves the task by combining a binary search and a simulated Stochastic Gradient Descent (SGD). The nodes are given a size in the visual space at each step in the binary search. Then, they are moved according to a SGD algorithm optimizing a stress function to remove existing overlaps, with a tolerance of at most 1 pixel in the VS. If the overlaps are solved, the next step in the binary search will make the nodes larger to maximize their size. If there remain overlaps, the nodes are made smaller to ease the overlap constraints.

Algorithm 1 GIST pseudo-code. **Blue** (resp. **green**) variables correspond to values in the **Geometric** (resp. **Visual**) space.

```
1:  Methods
2:  |  visualToGeometric(v, R, X): returns the value corresponding to v mapped from R's VS to X's GS
3:  |  geometricToVisual(v, R, X): inverse of visualToGeometric
4:  |  tolerance(D, R, X): returns the GS's tolerance such that it does not exceed 1 pixel and geometricToVisual(D−2∗t)> 1 in the VS
5:  |  overlaps(X, D, t): returns the set of overlapped node pairs with t tolerance
6:  |  moveNodes(X, O): move overlapped pairs of nodes to optimize stress by SGD [40]
7:  end Methods
8:
9:  Input Variables
10: |  X^0: Initial layout, position of every node in GS
11: |  D: Nodes' diameter
12: |  R: Target resolution
13: end Input Variables
14:
15: procedure GIST (X^0, D, R)
16: |  minDiam ← 1
17: |  maxDiam ← geometricToVisual(D, R, X)
18: |  stopBS ← false
19: |  X' ← X^0
20: |  while not stopBS do
21: |  |  diam ← (minDiam + maxDiam)/2
22: |  |  X ← X'
23: |  |  t ← tolerance(diam, R, X)
24: |  |  O ← overlaps(X, visualToGeometric(diam), t)
25: |  |  X ← moveNodes(X, O)
26: |  |  fail ← overlaps(X, visualToGeometric(diam), t) ≠ ∅
27: |  |  if fail then
28: |  |  |  maxDiam ← diam                    ▷ reduce next diameter
29: |  |  else
30: |  |  |  X' ← X                            ▷ save current solution
31: |  |  |  minDiam ← diam                    ▷ increase next diameter
32: |  |  if maxDiam - minDiam < ε then
33: |  |  |  if X' = X^0 then                  ▷ re-start with higher R
34: |  |  |  |  R ← R∗2
35: |  |  |  |  minDiam ← 1
36: |  |  |  |  maxDiam ← geometricToVisual(D, R, X)
37: |  |  |  else
38: |  |  |  |  stopBS ← true
39: |  return X'
```

3.1 Search for Optimal Node Diameters with Tolerance

GIST's input is a layout where every node has a position and diameter in the Geometric Space (GS). To produce a layout that guarantees all nodes remain visible (*i.e.*, with at least 1 pixel) in a given resolution R, it manages the layout in both the GS (for node movements optimization) and the Visual Space (VS) (to guarantee nodes' visibility). R is provided as a target resolution but can be modified during the optimization if it cannot lead to a solution. For instance, R is increased when $R^2 < N$ as there would not be enough space to draw all nodes with at least 1 pixel.

As presented in Algorithm 1, the main building block of GIST is a Binary Search (see line 20) for the optimal node diameters in the VS defined by R. That is to say, the algorithm tries to produce a layout where the nodes are as wide as possible in R and where all nodes are *almost* equally visible. We say *almost* equally visible, because the algorithm tolerates a few overlaps between the nodes as long as (i) this

overlap is not wider than 1 pixel in the VS of resolution R, and (ii) the node diameters remain above 1 pixel considering that tolerance. Doing so greatly helps the algorithm to converge, and guarantees that nodes are visible. The tolerance can be sub-pixel as it remains useful to the convergence once converted into the GS. Indeed, it would produce an overdraw-free visualization because of the image rendering rasterization, while relaxing the strictness of the overlap removal constraints in the GS. As defined in lines 24 and 26 of Algorithm 1, the tolerance is used when identifying overlaps. Two nodes i and j overlap if $d(X_i,X_j) < r_i + r_j - t$ where t is the tolerance.

At every step P in the Binary Search (*i.e.*, for a given diameter in the VS of resolution R), the overlapped nodes are moved to optimize a stress function (see line 25). The details of the node movements computation is postponed to the next Section 3.2. If there are still overlaps after the movements in the step P, then $P+1$ will try to solve the task with smaller nodes, which simplifies the task (see line 28). If there are no longer overlaps in step P, we save the current layout (line 30) and $P+1$ will be given a larger diameter (line 31). The algorithm therefore searches the largest diameter of the nodes representation that enables to solve the task.

Finally, if the difference between the bounds of the Binary Search is smaller than some threshold value ε (typically set to 10^{-3} in our experiments) the search is ended. At that point, if the task was not solved (line 33–36), we consider that it cannot be solved in the current resolution R and we re-start the algorithm with a higher one (here, doubled as shown in line 34). Restarting the optimization entirely is computationally expensive. However, it rarely happens in practice if the target resolution provided to the algorithm is not irrelevant in regard of the number of elements to draw. If the task was solved (see line 38), the search is ended and the adjusted layout is returned.

3.2 Node Movements

As defined in the literature, they combine the constraints of (i) removing overlaps and (ii) preserving the distances between non-overlapped nodes by modeling them into a stress function (see Equation 1). Let $O \in N \times N$ be the set of overlaps in a layout X, where $(i,j) \in O$ if nodes i and j overlap. Such a set of overlaps can be computed efficiently with the *scan-line* algorithm [7], where constraints are softened with some *tolerance* (see Section 3.1). The *ideal distance* to converge towards is then defined between every pair of nodes according to their belonging to O. As this paper considers circular node representations, the ideal distance between overlapped nodes is set to the sum of their radiuses, *i.e.*, the shortest distance such that they do not overlap anymore.

Node movements are computed to optimize this stress by Stochastic Gradient Descent [40]. This approach achieves state-of-the-art performance in stress optimization and gives the capability to bound the node movements to a *budget* by giving the algorithm a fixed number of iterations to converge. However, the approach does not scale well as it optimizes $\mathcal{O}(N^2)$ distances. To alleviate this, we only optimize distances between overlapped nodes in GIST. The ideal distance and optimized stress can be formalized as:

$$\delta_{ij} = r_i + r_j, (i,j) \in O \tag{2}$$

$$\sigma(X) = \sum_{(i,j)\in O} W_{ij} \left(d\left(X_i,X_j\right) - \delta_{ij}\right)^2 \tag{3}$$

where W_{ij} is set to δ_{ij}^{-2}. As defined in [40], the optimization is achieved by moving the nodes in direction of the stress' gradient.

4 Evaluation

This section presents the quantitative evaluation of GIST, as well as its comparison with state-of-the-art algorithms.

4.1 Evaluation protocol

4.1.1 Metrics

The quality metrics considered in this evaluation are taken or inspired from Li *et al.* [24] and Chen *et al.* [3, 4]. We made adjustments to some selected metrics to better reflect their purpose. We also designed two metrics to measure upscaling effects and node visibility in a fixed resolution as these aspects are of uttermost importance and were not evaluated in previous works.

Normalized node movements This metric inspired from Chen *et al.* [3, 4] quantifies the relative node movements made from the original layout to the adjusted one. Before computing movement, a scale and shift operation *superimposes* the adjusted layout on the original one and not the opposite as proposed in [3, 4]). It enables the movement quantity to be unsensitive to offsets and movements induced by upscaling effects. Hence, it makes the comparison of various adjusted layouts possible as the relative node movements are always quantified in the original layout's space. The result is normalized by the highest movement possible: the length of the layout's rectangular bounding box diagonal. The scale operation ensures that both bounding boxes are of corresponding size (preserving the adjusted layout aspect ratio), while the shift operation centers both bounding boxes on the same location. Formally, the metric can be defined as:

$$NNM(X',X^0) = \frac{\frac{1}{N}\sum_{i=1}^{N} \mathrm{d}(X_i^0, \mathrm{shift}(\mathrm{scale}(X_i')))}{\sqrt{BB_{width}^2 + BB_{height}^2}} \tag{4}$$

where X' is the adjusted layout and BB_{width} (resp. BB_{height}) is the width (resp. the height) of the original layout's bounding box.

Shape preservation Inspired by Li *et al.* [24], this metric measures the preservation of the layout's global shape. We draw several concentric circles of increasing radius, centered on the initial layout's center of mass, noted $\mu(X)$. We uniformly pick 36 points $\left\{p_{i,k}, 1 \le k \le 36\right\}$ on each circle c_i. For each point $p_{i,k}$ of each circle c_i, we choose the closest node $X_{i,k}$ in the initial layout except if $d(X_{i,k},c_i) > T$ where T is a threshold typically set to 10 as proposed in [24]. In that case, no node is associated to $p_{i,k}$ and a circle C_i will be ignored if it only gathers 0 or 1 node. For each chosen node $X_{i,k}$, we compute the inverse ratio of the distance between $X_{i,k}$ and the center of mass $l_{i,k} = \mathrm{d}(X_{i,k},\mu(X))$ and the analog of that distance in the result layout $l'_{i,k} = \mathrm{d}(X'_{i,k},\mu(X'))$. We then compute Shape preservation as the average of the distance ratio's coefficient of variation across all the circles.

$$SP(X',X^0) = \frac{1}{C}\sum_{i=1}^{C} \frac{std\left(\left\{\frac{l'_{i,k}}{l_{i,k}}, X_{i,k} \in C_i\right\}\right)}{\mu\left(\left\{\frac{l'_{i,k}}{l_{i,k}}, X_{i,k} \in C_i\right\}\right)} \tag{5}$$

where C is the number of circles initially set to 20, but can be smaller if some circles are discarded because they do not gather at least 2 nodes.

Ordering similarity Taken from Li *et al.* [24], it measures the preservation of relative node ordering from different angles. An axis with a given angle is drawn on the original layout, and every node's position is orthogonally projected on it. By repeating the process with the same axis on the adjusted layout, we obtain two sequences of nodes alongside the axis. A similarity score is then given to the axis with a Kendall correlation coefficient [21]. In our evaluation, we repeated the process for 30 angles with regular intervals [24], and the score of the metric is the average of the axes similarity scores.

K-neighborhood preservation Captures the stability of the neighborhood. It averages, for each node, the portion of the k-nearest neighbor set of the node which still belongs in the k-nearest neighborhood set in the result layout.

$$KNP(X',X^0) = \frac{1}{N}\sum_{i=1}^{N} \frac{|\mathrm{KNN}(X_i,k) \cap \mathrm{KNN}'(X_i',k)|}{k} \tag{6}$$

where KNN (resp. KNN$'$) returns the k-neighborhood of a node in the original (resp. adjusted) layout. Considering the distance d between the node of interest and the k^{th} closest neighbor, there may be more than k nodes at distance less than d. To overcome that issue, we chose in both layouts the sets with the largest intersection. In this article, k is set to 10 as proposed in [24].

Density preservation Taken from [24], this metric measures the preservation of node density in the layout. Each node is associated with a local density measure using a KNN set. The local density measure is set as $\rho_i = \frac{1}{\frac{1}{k}\sum_{j \in \text{KNN}_i} \text{d}(X_i, X_j)}$. The nodes are sorted by their local density measure and associated with their quantile q_i in that sorted order. The metric's score is the average (across the nodes) of the differences between their quantile in the original and the adjusted layout.

$$DP(X', X^0) = \frac{1}{N}\sum_{i=1}^{N} |q_i - q'_i| \tag{7}$$

Scaling This metric measures the preservation of the original layout's scale. The global density score of a layout is computed as the ratio between its nodes' cumulated area over its bounding box area. The metric is then defined as the ratio of the density score in the adjusted layout on that of the initial layout.

$$Scale(X', X^0) = \frac{\sum_{i=1}^{N} {r'_i}^2 * \pi}{BB'_{area}} \Big/ \frac{\sum_{i=1}^{N} r_i^2 * \pi}{BB_{area}} \tag{8}$$

where r_i (resp. r'_i) is the radius of node i in the original (resp. adjusted) layout, and BB_{area} (resp. BB'_{area}) is the area of the original (resp. adjusted) layout's rectangular bounding box.

Nodes' mininimal number of visible pixels This metric represents the number of pixels occupied solely by the least visible node in the adjusted layout when rendered in a resolution $R \times R$ (set to 2000×2000 pixels in our evaluation) :

$$NMVP(X', X^0) = \min_{1 \leq i \leq N} |P_i| \tag{9}$$

where $P_i = \left\{ p_{j,k}, 1 \leq j,k \leq R, n_i \text{ is the only node in pixel } p_{j,k} \right\}$.

4.1.2 Dataset

The dataset of this evaluation is composed of 48 scatterplots from two sources. First, we borrow the data of the ScatterplotUnfold [24] article. Because some algorithms did not scale well to very large scatterplots, they executed them on sub-samples of size $N = 3000$. However, as we aim at executing all the algorithms on the same datasets in our evaluation, we only conserve the 38 scatterplots from [24] where $N < 150000$ such that all the benchmarked algorithms can be evaluated on them. To counterbalance the samples' removal, we added 10 datasets with $N \in [1024; 58509]$ from the Multidimensional Scaling and Machine Learning fields [23, 36, 39] projected in 2D using t-SNE [35]. Figure 2 presents the number of overlaps against the number of nodes in the 48 selected samples. In all the scatterplots, the initial node diameters provided to the Layout Adjustment algorithms is $D = 2$. The positions and diameters in these original layouts are considered to be in the Geometric space as they were not generated for any specific resolution.

4.2 Quantitative Evaluation

Settings. In this quantitative study, GIST is compared with state-of the art Layout Adjustment algorithms. Namely, the selected algorithms are: ScatterplotUnfold (SU) [24], DGrid [15], HaGrid [6], PFS' [14], GTree [30] and PRISM [9], and were presented in Section 2.

All the algorithms were executed with fixed parameters across the 48 scatterplots presented in Section 4.1.2. The parameters settings for every algorithm and metric are reported in the Supplementary Materials. For the evaluation, we used our own implementation of GIST (our algorithm) and PFS'. ScatterplotUnfold implementation is taken from their own repository[1] while DGRID and HaGrid are taken from HaGrid's one[2]. These 5 algorithms were executed on an Intel Core i9-12900KF CPU. PRISM and GTree implementations are taken from the Microsoft Automatic Graph Layout (MSAGL) library [31] and were executed on an Intel Core i7-9700K. We do not include FORBID [10] algorithm in this evaluation as it did not complete on scatterplots with $N > 50000$ (out of memory), and produced overlap-free layouts with too much nodes dispersion when it did complete.

[1] www.github.com/diyike/scatterplotUnfold

[2] www.github.com/saehm/hagrid

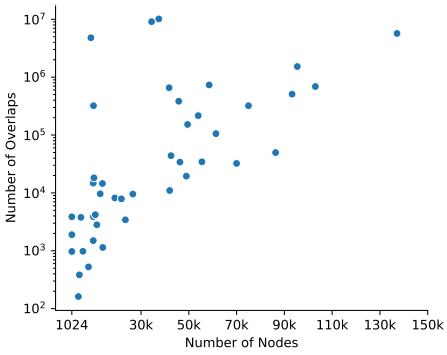

Fig. 2: Number of overlaps against the number of nodes in the 48 scatterplots. Y axis has a log scale. One scatterplot with $N = 10000$ is not represented as it does not have any overlap.

We conducted an additional evaluation adding a tolerance (*i.e.*, negative padding) in the GS for the Node Dispersion algorithms PFS', PRISM and GTree. Its results, available in the Supplementary Materials, demonstrated that the benefits are not significant. Hence, the padding for these algorithms was set to 0 in the next evaluation.

The resulting layouts of these methods are evaluated according to the metrics defined in Section 4.1.1. The metric scores are reported in Figure 3 and discussed in the next. The results are studied under 3 criteria: initial layout preservation, nodes visibility and execution time.

Minimizing initial layout's distortion. To measure how well a resulting layout has preserved the initial layout, we focus on the 5 first metrics of Figure 3. One can see that GIST outperforms state-of-the-art algorithms on Node Movements minimization (3a), Shape preservation (3b) and Ordering similarity preservation (3c). These metrics capture the global preservation of the initial layout, demonstrating GIST capability in that criterion. The difference is smaller with ScatterplotUnfold (SU) on Ordering similarity and Node Movements, but overall the results are in favor of GIST.

On Density (3e) and K-Neighborhood (3d) preservation, GIST median performance is at the same level as SU, but has higher variations. These metrics are more dedicated to quantifying preservation of local neighborhood in the scatterplots. We explain the high variations of GIST by the combination of the binary search for the optimal node diameters and the increase in the resolution if the binary search does not end with a correct solution (see Section 3). On some scatterplots, the best solution is found at the end of the binary search, meaning that many movements have been cumulated to obtain the solution layout. On the other hand, if a solution is found in the early stages of a binary search (either the first one, or after increasing the target resolution), the local neighborhoods are better preserved.

Overall, HaGrid and DGrid underperformed and do not optimize the preservation of the initial layout criterion well. This behavior could be expected as both are meant to produce more compact visualizations, maximizing the nodes' visibility; as we will see in the Section 5.1. PFS', GTree and PRISM results on this criterion are mitigated. None of them particularly stands out and their ranking does not seem to follow any trend across the metrics. GIST competes with SU on the initial layout preservation criterion. It is the best method to preserve the global shape of the layouts, and among the best to preserve their local structures. Figure 4 illustrates the relative node movements produced by GIST. Each line represents a node's movement from its position in the original layout to its relative position in the adjusted layout. These examples demonstrate how the computed node movements preserve the original's inner and outer structures.

Maximizing nodes visibility. We evaluate this criterion in regard of the Scaling (3f) and Nodes' min. visible pixels (3g) metrics. As

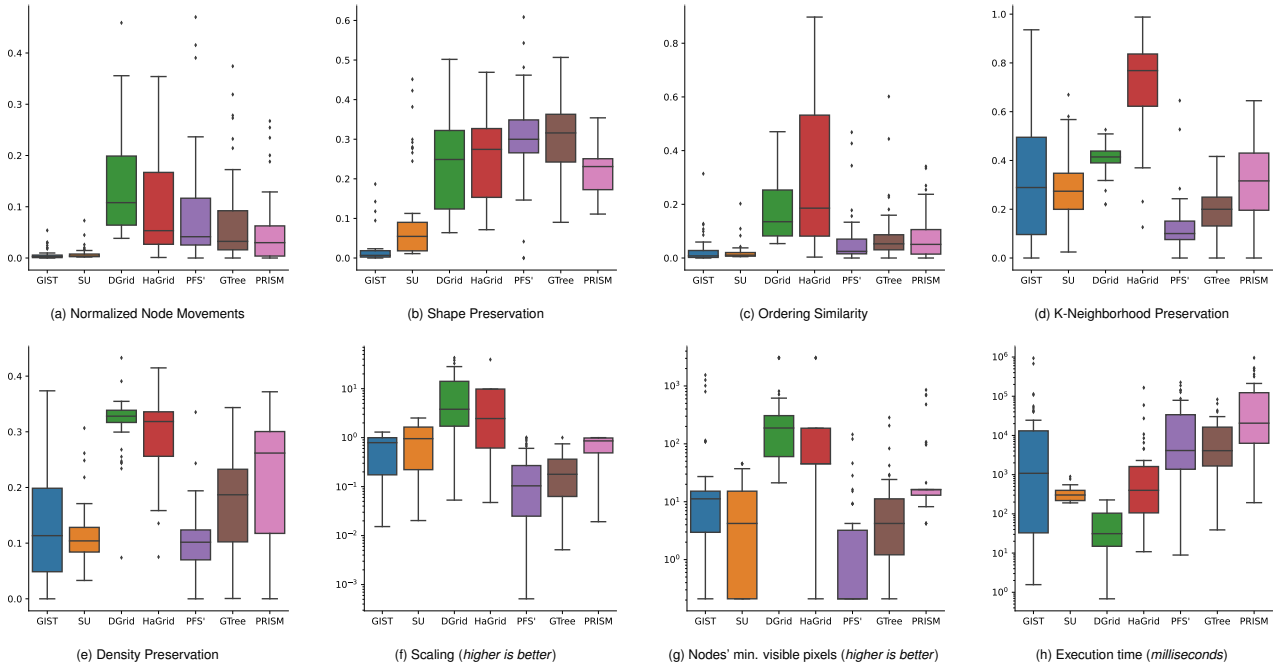

Fig. 3: Metric score distributions across the 48 samples of our dataset. All metrics are oriented *lower is better*, except (f) and (g). The three last plots have log scales on the Y axis. The metrics in (a), (b), and (c) capture the preservation of the global shape of the original layout. (d), and (e) focus on the preservation of local structures of the original layout. Eventually, (f) and (g) enable to apprehend how visible the nodes are in the adjusted layouts.

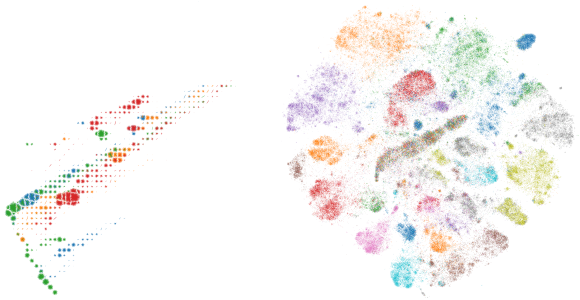

Fig. 4: Relative node movements in two layouts produced by GIST. The adjusted layout is superimposed (shifted and rescaled) on the original layout, and a line is drawn for each node between its position in the original layout, and its position in the adjusted one.

opposed to the previously studied metrics, they are oriented *higher is better* and capture two aspects of the nodes' visibility maximization.

Scaling measures how the nodes' visibility in the produced layout (based on the node areas and the layout bounding box) is representative of their visibility in the original layout. The scores can be above one if an adjusted layout is more compact than the original one. As expected, this is especially observable with DGrid and HaGrid since they gridify the original layout to produce compact visualizations. GIST's Scaling score is close to that of SU and PRISM. All three have Scaling scores close to one. In other words, they produce adjusted layouts where the nodes are as visible as in the original layout, demonstrating that they leverage empty spaces to resolve overlaps.

Nodes' min. visible pixels score represents the number of pixels exclusively reserved to the least visible node(s) in a 2000×2000 resolution for each scatterplot. Making sure that the least visible node is represented with $p_{min} = 1$ pixel means that all the nodes are visible, and maximizing p_{min} is preferable to improve the layout's readability in that resolution. Again, DGrid and HaGrid compact visualizations have the widest least visible node(s). As expected, most layouts produced by GIST lead to least visible nodes of at least 1 pixel. The only exceptions are the scatterplots where GIST was not able to find in a solution in the metric's resolution $R = 2000$. This happened 8 times for GIST, 14 for SU, 4 for HaGrid, 28 for PFS' and 10 for GTree. SU and PFS' also failed on the 8 problematic cases for GIST, while GTree succeeded for one of them. DGrid, HaGrid and PRISM successfully provided an overlapfree layout with $p_{min} > 1$, but at the cost of severe distortions. Despite these problematic cases, GIST's least visible nodes' distribution show that $p_{min} > 1$ in most cases. SU and GTree results are slightly under GIST's ones, and PFS' produced layouts with the smallest least visible nodes.

Time costs. Since GIST backbone leverages FORBID [10], its complexity inherits from FORBID's one in $\mathcal{O}(s(N^2 + N\log(N))$ where s is the depth of the binary search. The N^2 complexity comes from the optimization of distance between all pairs of nodes. Although GIST alleviates this heavy of the complexity by only optimizing distances between the overlapped nodes, its worst case complexity remains quadratic as the number of overlaps is bounded by N^2. Hence, GIST complexity is in $\mathcal{O}(s(|O| + N\log(N))$ where $|O|$ is the number of overlaps.

Nevertheless, Figure 3h shows that GIST execution time is better than other quadratic state-of-the-art algorithms (*i.e.*, PFS', GTree and PRISM). However, it loses to SU, DGrid and HaGrid who all have linearithmic complexity. The problematic cases for GIST are scatterplots where the nodes' distribution is very dense in specific regions while others are almost empty. Figure 5 top row illustrates the problematic. In this example, GIST's solution was found with a resolution of $R = 4000$. Since GIST tries to minimize the original layout's distortion, it is only allowed to move the nodes with a *budget* of movements. If the overlapped nodes are located in a small and dense region, the budget of movements is not sufficient to move the nodes far enough to leverage the layout's empty spaces. This problematic was already identified by Li *et al.* [24] and many Layout Adjustment algorithms by Node Dispersion are affected. On the other hand, GIST is very efficient (and sometimes faster than SU) on scatterplots where the nodes are more uniformly distributed across the original layout's bounding box. An example of *easy* case is presented in Figure 5 bottom row. In practice, we did not observe problematic cases often, as the data we consider are mostly organized in spread out clusters. They appeared when there is a small dense area in the initial layout,

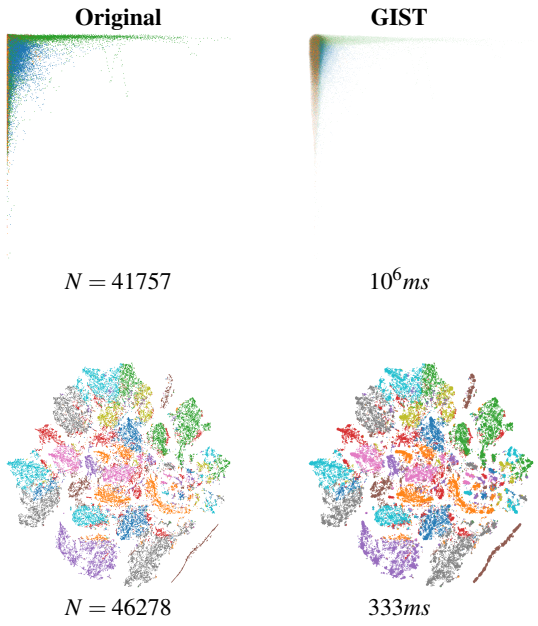

Fig. 5: A *problematic* (top) and an *easy* (bottom) case for GIST. While both have similar sizes ($N = 41757$ against $N = 46278$), the algorithm is much faster ($333ms$) on the bottom example than on the top one ($10^6 ms$).

that is *confined* in a zone of the layout's bounding box because of some sparser areas. Since many methods aim at preserving the layout's global shape, such distribution of points is problematic. We believe that it is possible to anticipate such problematic cases by measuring the distribution of distances or densities in the representation. For instance, the Sparsity [12] score of the scatterplots in the Figure 5 are 894 for the top row against 146 for the bottom one.

Overall, we identified three categories of algorithms. (i) Methods such as GIST and SU are best suited to global shape preservation. These two methods compete with (ii) standard Node Dispersion (ND) algorithms such as PFS' and GTRee, PRISM. Overall, our results corroborate those of Chen *et al.* [3,4] survey on ND algorithms: PFS' is better than PRISM and GTree on local neighborhood preservation, but worse on global shape preservation. Finally, (iii) DGrid and HaGrid win on node visibility maximization as they produce compact visualizations. Although they are also the fastest, they severely distort the initial layout. Hence, they should only be preferred when compactness is a high priority. On the other hand, we showed that approaches such as GIST and SU are preferable to standard ND algorithms to preserve the global shape of the initial layout, while also retaining a good level of local neighborhood preservation. They also increase the the nodes visibility in comparison to ND approaches, and scale better to large data. Eventually, GIST provides better compromise between layout preservation and nodes visibility than SU on most cases.

5 Discussion

In this section, we compare GIST and state-of-the-art algorithms on some visual examples. In addition, we qualitatively study the impacts of tuning the tolerance to overlaps in the visual space, and its relation with the target resolution. Finally we discuss a limitation of GIST.

5.1 Qualitative Evaluation

Figure 6 presents the adjusted layouts produced by GIST and all the state-of-the-art algorithms considered in the quantitative evaluation (see Section 4.2). As a reminder, an interactive demonstration of GIST is also available online [11]. The scatterplots presented cover a wide range of number of nodes (*i.e.*, from 1024 to 93239). In addition, they were selected as they do not have the problematic cases identified earlier. As a reminder, the *problematic* scatterplots can be defined as having some significantly small and dense regions, as illustrated in Figure 5. Since state-of-the-art algorithms are also affected by these distributions, they are not discussed here.

News popularity. One can see that *News popularity* was not properly drawn by either PFS' or GTree. Both have significantly reduced the node sizes on screen and the original layout structures are barely recognizable. Even though PRISM's layout is slightly better, it is not satisfactory in regard to the results of the remaining algorithms. DGRID has produced a compact visualization where all the nodes are visible. If we can identify the *blue* structures of the original layout, most patterns were lost during the layout adjustment process. For the three remaining algorithms, we consider that the produced layouts have successfully preserved the original layout's global and local patterns. Among them, GIST's layout have the most visible nodes (*i.e.*, the widest), followed by HaGrid then SU.

MoCap and GaAsH6. As opposed to *News popularity*, all the algorithms gave a relatively interpretable result on *MoCap* and *GaAsH6*. PFS' nodes are very small but the layout is recognizable despite having some distortions. GTree's layouts are not particularly satisfactory either on these examples. On the other hand, PRISM has produced a very compact layout that severely distorted *MoCap*. However, its result on *GaAsH6* is surprisingly satisfactory as the nodes are wider than other algorithms, and the layout patterns are preserved. One can see that HaGrid induces large deformations in dense regions of the layout in both examples. The same observation can be made for DGRID. Its compact results are not inconvenient on *GaAsH6* as the clusters are well defined and separated. However, they make *MoCap* difficult to read because nodes with the same label are distributed all across the layout, creating an effect of disorder. Finally, Both GIST and SU have preserved the original layout structures, and we consider GIST's results more satisfying as nodes are wider and the clusters are easier to identify.

Elec board This scatterplot illustrates one of the limitations of compact methods. Looking at DGRID and HaGrid layouts, it becomes obvious that even though this approach can produce satisfactory results, it cannot be used without additional visual encoding (*e.g.*, node colors) to identify the data structures. PRISM and GTree have seriously damaged the original layout. Although PFS' has preserved the original layout's patterns, the node sizes are too small in comparison with other algorithms. Again, GIST and SU outperform the other algorithms, with GIST proposing a layout where the nodes are wider.

Fashion and Helix On the two last scatterplots, PRISM has critically damaged the original layout. Although GTree, PFS', and SU have successfully preserved the original layout, they have significantly reduced the node sizes to a point that is not satisfactory. DGRID layouts are satisfactory on both scatterplots, although the "gridification" of *Helix* might not be necessary considering its size ($N = 1024$). It can even mislead a user by suggesting that some clusters are neighbors (*e.g.*, pink and red, brown and red) while they're not. While HaGrid result is great on *Fashion*, it suffers from a major deformation on *Helix*. In addition to the misleading effect on the clusters' neighborhood, the pink and blue clusters are mixed. Finally, GIST results on these two examples are arguably the best. They both retain the original layout's structures and maximize the nodes' visibility.

Overall, we observe that PRISM, GTree and PFS' do not provide satisfactory results on large scale scatterplots, even though we excluded those with problematic nodes' distribution from this qualitative evaluation. DGRID and HaGrid results are good to obtain a compact visualization but have several limitations. If there is no visual encoding to dissociate the nodes or the clusters, no information can be read out of their layout. Finally, their interpretation can be misleading as they suggest a proximity that does not exist between some nodes or clusters. Eventually, GIST proposes a great optimization of initial layout preservation and the nodes' visibility.

5.2 Increasing the Tolerance, Decreasing the Resolution

As presented in Section 3 and Algorithm 1, the *tolerance* to overlaps was set to 1 pixel in a Visual Space (VS) of resolution $R = 2000$. This

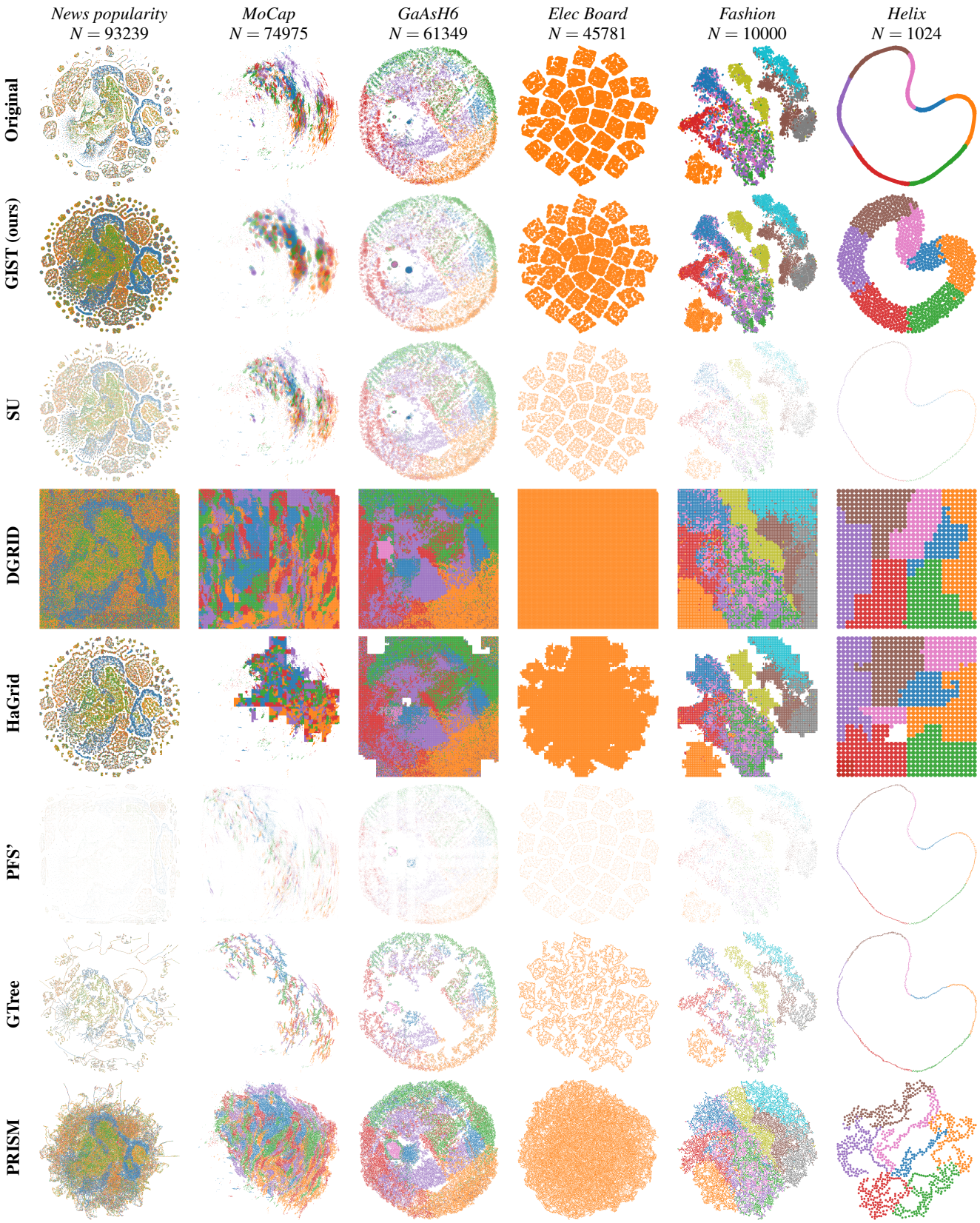

Fig. 6: Examples of adjusted layouts produced by GIST, ScatterplotUnfold (SU), DGRID, HaGrid, PFS', GTree and PRISM. The images have a resolution of 2000×2000px without anti-aliasing and nodes have an opacity of 90%. GIST found a solution in $R = 2000$ for all the scatterplots except *MoCap* (where $R = 4000$). They were rendered with the Matplotlib Python library [17] relying on the C++ rendering library Anti-Grained Geometry. It is recommended to read the figure with a viewer that can zoom (up to 6000%).

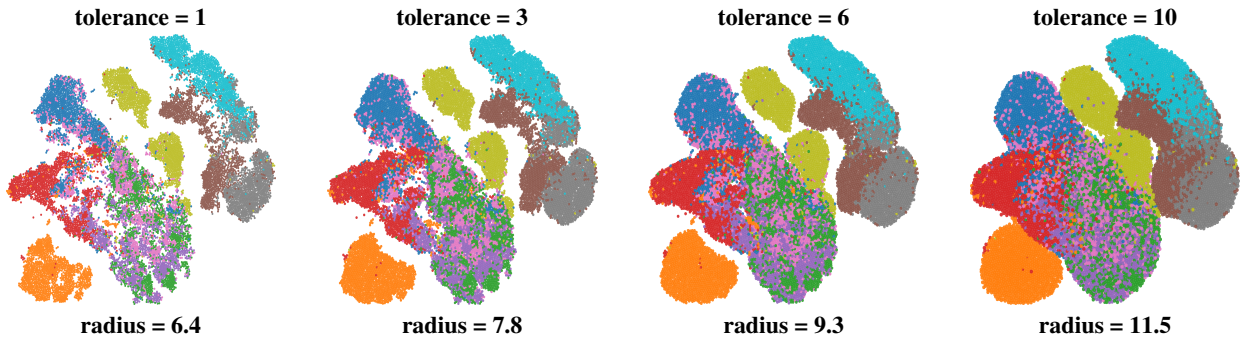

Fig. 7: GIST result layouts on *tsne-fashion* scatterplot with various tolerances. The target resolution is set to $R = 2000$ and node radiuses in that visual space (VS) are reported below each figure. Radiuses are floating pixel numbers computed with the interval change from the GS to the VS in resolution R. The rasterization is done *a posteriori* by the image generation.

choice seemed to be evident as it eases the algorithm's convergence while being harmless to the visualization in the VS.

Yet, it is trivial to support a higher tolerance, if one wants to produce an even more compact adjusted layout. However, it is important to make sure that the nodes always have at least 1 pixel exclusively dedicated to their representation. Formally, $D - 2 * t > 1$ where D and t are respectively the node diameters and the tolerance in the VS. The main goals of using a higher tolerance are: (i) produce a more compact layout; and (ii) fasten the algorithm's convergence.

Figure 7 presents GIST adjusted layouts of the *Fashion* scatterplot with visual tolerances $t \in \{1,3,6,10\}$ in a resolution 2000×2000. Below every layout is reported its corresponding node radiuses. These figures show that increasing the tolerance is not necessarily beneficial. As we aim at guaranteeing the nodes' visibility, we always consider the worst case scenario. Hence, when the node radius is 6.4 with a tolerance of 1, we consider that the nodes will be represented with a radius of $r_{min} = 6.4 - 1$ pixels. With the image rasterization, we can consider the worst case where the nodes will only be displayed with $r_{min} = \text{floor}(5.4) = 5$ pixels. With the same reasoning, we observe that the minimum number of exclusive pixels we can guarantee decreases $r_{min} \in \{5,4,3,1\}$ as the tolerance increases $t = \{1,3,6,10\}$. Although it does not aesthetically seem beneficial to increase the tolerance, it can be very useful to improve the reactivity in interactive visualizations scenarios (see GIST's demo [11]).

The target visual space resolution R was fixed during the experiment to $R = 2000$. The effects of varying this parameter while setting the *tolerance* to a fixed value is the same as doing the opposite. In fact, the combination of *tolerance* and R define the available empty spaces that the algorithm can use to either move or grow the nodes. Having a tolerance of $t = 1$ pixel with $R = 1000$ allows as much available space as having $t = 2$ and $R = 2000$ (*i.e.*, nodes are larger but can overlap more). It then leads to the same result as long as the image renderer's rasterization is uniform. Visual examples with varying R are proposed in the Supplementary Materials. In practice, we expect R to be fixed depending on the use case, while t has to be chosen carefully.

5.3 Limitation

The main limitation that was not yet discussed is GIST's behavior when the binary search ends while no solution was found. This mostly happens on complex scatterplots where the algorithm hardly converges towards a solution, and is already computationally expensive. But what is not satisfactory is the need to restart the optimization from the beginning with a higher resolution. This comes down to call GIST several times on the same input by changing the target resolution, and is severely expensive. Pre-processing based on the size of the original layout could enable to preemptively refine the target resolution.

Although this behavior can be perceived as a limitation in terms of computational cost, it is the result of a design choice we made to ensure that GIST provides a layout that optimizes the initial layout preservation. The other choice would have been to let the algorithm move the nodes until it eventually converges. However, such a behavior would significantly distort the original layout. There are already several Layout Adjustment algorithms that allow severe distortions of the original layout to produce compact embeddings, and we believe that GIST proposes a good alternative with a unique compromise between nodes' visibility and initial layout preservation.

6 Conclusion

2D scatterplots are nowadays a common way to visualize high dimensional datasets. However, both the 2D projection and the drawing of the nodes with a non-zero width shape generate occlusions that hinder the representation. Such an issue is commonly solved by using an overlap removal algorithm that guarantees that all occlusions have been removed, at the cost of deformations of the original data.

In this article, we proposed GIST: a novel method that can be parametrized to perfectly or partially solve the occlusion issues. As such, we prefer to use the term layout adjustment rather than overlap removal as the user can accept partial occlusions. It relies on an hybrid method that both operates in the Geometric Space, by moving node coordinates thanks to the optimization of a stress function by gradient descent, and in the Visual Space, by choosing the right nodes size using a binary search. The algorithm's goal is to optimize three criteria: (i) nodes visibility guarantee (at least 1 pixel), (ii) node size maximization, and (iii) the original layout preservation. By design, it is able to handle larger datasets (in term of number of points) than other GS baseline methods. An interactive demo of GIST is available [11].

An extensive evaluation has shown the efficiency of GIST on various criteria. It has been compared to 6 state-of-the-art methods against 48 datasets of various complexity. It competes well against baseline methods, and is better suited in several scenarios (*e.g.*, cluster shape preservation, outlier visibility, distribution dis-ambiguity).

The evaluation has also shown some limitations that could be tackled in future works. For example, the algorithm has shown to be sensitive to the distribution of node densities in the original layout. These cases were also problematic for the other algorithms and it would be interesting to investigate how to alleviate such problematic distributions. In the Geometric Space, one could use some spatial deformation according to local data densities. This would allow the nodes in dense regions to move more, while keeping the movements allowed for those in sparse regions reasonable. An other lead for future work is to optimize other visual encodings in GIST. For example, the notion of *tolerance* to overlaps could also include the optimization of the nodes *opacity*. This could enable to have a higher tolerance (in number of pixels) while still optimizing node visibility. The nodes rendering order could also be optimized to maximize the visible node areas considering they can partially overlap. To optimize perceptual metrics [27] is the next step towards efficient human-centered visualization as it will enable to go beyond the guarantee of visibility by ensuring the nodes *perceivability* (*i.e.*, make sure that the human perception system is able to capture and faithfully interpret data representation).

Acknowledgments

This research was founded by the french ANR project InvolvD.

Supplementary Materials

The suppl. materials provided alongside this manuscript contain:

- A video simulating steps in the GIST execution to illustrate its convergence (this does not represent the real speed);
- Metrics and baseline Algorithms parameter settings;
- A study on adding *negative padding* to the Node Dispersion algorithms PFS', PRISM and GTree;
- Visual examples illustrating the similar effect between increasing tolerance (with a fixed resolution) and reducing the resolution (with a fixed tolerance).

References

[1] E. Bertini and G. Santucci. By chance is not enough: preserving relative density through nonuniform sampling. In *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.*, pp. 622–629. IEEE, 2004. doi: 10.1109/IV.2004.1320207 2

[2] U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *International Symposium on Graph Drawing*, pp. 218–229. Springer, 2008. doi: 10.1007/978-3-642-00219-9_21 2

[3] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry. Node Overlap Removal Algorithms: A Comparative Study. In *International Symposium on Graph Drawing and Network Visualization*, pp. 179–192. Springer, 2019. doi: 10.1007/978-3-030-35802-0_14 2, 4, 7

[4] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry. Node Overlap Removal Algorithms: an Extended Comparative Study. *Journal of Graph Algorithms and Applications*, 24(4):683–706, 2020. doi: 10.7155/jgaa.00532 2, 4, 7

[5] X. Chen, T. Ge, J. Zhang, B. Chen, C.-W. Fu, O. Deussen, and Y. Wang. A Recursive Subdivision Technique for Sampling Multi-class Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):729–738, 2020. doi: 10.1109/TVCG.2019.2934541 2

[6] R. Cutura, C. Morariu, Z. Cheng, Y. Wang, D. Weiskopf, and M. Sedlmair. Hagrid — Gridify Scatterplots with Hilbert and Gosper Curves. In *Proceedings of the 14th International Symposium on Visual Information Communication and Interaction*, pp. 1–8. Association for Computing Machinery, 2021. doi: 10.1145/3481549.3481569 2, 3, 5

[7] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast Node Overlap Removal. In *International Symposium on Graph Drawing*, pp. 153–164. Springer, 2005. doi: 10.1007/11618058_15 2, 4

[8] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast Node Overlap Removal—Correction. In *International Symposium on Graph Drawing*, pp. 446–447. Springer, 2007. doi: 10.1007/978-3-540-70904-6_44 2

[9] E. Gansner and Y. Hu. Efficient, Proximity-Preserving Node Overlap Removal. *Journal of Graph Algorithms and Applications*, 14(1):53–74, 2010. doi: 10.7155/jgaa.00198 2, 5

[10] L. Giovannangeli, F. Lalanne, R. Giot, and R. Bourqui. FORBID: Fast Overlap Removal By stochastic gradIent Descent for Graph Drawing. In *Graph Drawing and Network Visualization*, pp. 61–76. Springer, 2023. doi: 10.1007/978-3-031-22203-0_6 2, 5, 6

[11] L. Giovannangeli, F. Lalanne, R. Giot, and R. Bourqui. Guaranteed Visibility in Scatterplots with Tolerance. `https://labribkb.github.io/gist/`, 2023. 2, 7, 9

[12] T. Gomez, T. Fréour, and H. Mouchère. Metrics for Saliency Map Evaluation of Deep Learning Explanation Methods. In *Pattern Recognition and Artificial Intelligence*, pp. 84–95. Springer International Publishing, 2022. doi: 10.1007/978-3-031-09037-0_8 7

[13] A. Halnaut, R. Giot, R. Bourqui, and D. Auber. VRGrid: Efficient Transformation of 2D Data into Pixel Grid Layout. In *26th International Conference Information Visualisation (IV)*, pp. 11–20. IEEE, 2022. doi: 10.1109/IV56949.2022.00012 3

[14] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara. A Layout Adjustment Problem for Disjoint Rectangles Preserving Orthogonal Order. In *Graph Drawing*, pp. 183–197. Springer, 1998. doi: 10.1007/3-540-37623-2_14 2, 5

[15] G. M. Hilasaca, W. E. Marcílio-Jr, D. M. Eler, R. M. Martins, and F. V. Paulovich. A Grid-based Method for Removing Overlaps of Dimensionality Reduction Scatterplot Layouts. *arXiv preprint arXiv:1903.06262*, 2023. doi: 10.48550/arXiv.1903.06262 2, 3, 5

[16] X. Huang, W. Lai, A. S. M. Sajeev, and J. Gao. A new algorithm for removing node overlapping in graph visualization. *Information Sciences*, 177(14):2821–2844, 2007. doi: 10.1016/j.ins.2007.02.016 2

[17] J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.558

[18] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau. ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018. doi: 10.1109/TVCG.2017.2744718 1

[19] M. Kahng, N. Thorat, D. H. Chau, F. B. Viégas, and M. Wattenberg. GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):310–320, 2019. doi: 10.1109/TVCG.2018.2864500 1

[20] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. doi: 10.1016/0020-0190(89)90102-6 2

[21] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. doi: 10.2307/2332226 4

[22] J. B. Kruskal and M. Wish. *Multidimensional scaling*, vol. 11. Sage, 1978. doi: 10.4135/9781412985130 2

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791 5

[24] Z. Li, R. Shi, Y. Liu, S. Long, Z. Guo, S. Jia, and J. Zhang. Dual Space Coupling Model Guided Overlap-Free Scatterplot. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):657–667, 2023. doi: 10.1109/TVCG.2022.3209459 2, 3, 4, 5, 6

[25] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426*, 2018. doi: 10.48550/arXiv.1802.03426 1

[26] W. Meulemans. Efficient Optimal Overlap Removal: Algorithms and Experiments. *Computer Graphics Forum*, 38(3):713–723, 2019. doi: 10.1111/cgf.13722 2

[27] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards Perceptual Optimization of the Visual Design of Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 23(6):1588–1599, 2017. doi: 10.1109/TVCG.2017.2674978 9

[28] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu. Understanding Hidden Memories of Recurrent Neural Networks. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 13–24. IEEE, 2017. doi: 10.1109/VAST.2017.8585721 1

[29] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages & Computing*, 6(2):183–210, 1995. doi: 10.1006/jvlc.1995.1010 2

[30] L. Nachmanson, A. Nocaj, S. Bereg, L. Zhang, and A. Holroyd. Node Overlap Removal by Growing a Tree. In *Graph Drawing and Network Visualization*, pp. 33–43. Springer, 2016. doi: 10.1007/978-3-319-50106-2_3 2, 5

[31] L. Nachmanson, S. Pupyrev, T. Dwyer, and T. Hart. Microsoft Automatic Graph Layout. `https://www.microsoft.com/en-us/research/project/microsoft-automatic-graph-layout/`, 2007. 5

[32] M. Ortmann, M. Klimenta, and U. Brandes. A Sparse Stress Model. In *Graph Drawing and Network Visualization*, pp. 18–32. Springer, 2016. doi: 10.1007/978-3-319-50106-2_2 2

[33] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen. Rolled-out Wordles: A Heuristic Method for Overlap Removal of 2D Data Representatives. *Computer Graphics Forum*, 31(3):1135–1144, 2012. doi: 10.1111/j.1467-8659.2012.03106.x 2

[34] G. Strong and M. Gong. Self-Sorting Map: An Efficient Algorithm for Presenting Multimedia Data in Structured Layouts. *IEEE Transactions on Multimedia*, 16(4):1045–1058, 2014. doi: 10.1109/TMM.2014.2306183 3

[35] L. Van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. 1, 5

[36] L. Van Der Maaten, E. Postma, J. Van den Herik, et al. Dimensionality Reduction: A Comparative Review. Technical report, Tilburg University Technical Report, 2009. 5

[37] J. Wang, L. Gou, H. Yang, and H.-W. Shen. GANViz: A Visual Analytics Approach to Understand the Adversarial Game. *IEEE Transactions on*

Visualization and Computer Graphics, 24(6):1905–1917, 2018. doi: 10.1109/TVCG.2018.2816223 1

[38] L.-Y. Wei. Multi-Class Blue Noise Sampling. *ACM Transactions on Graphics*, 29(4), 2010. doi: 10.1145/1778765.1778816 2

[39] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017. doi: 10.48550/arXiv.1708.07747 5

[40] J. X. Zheng, S. Pawar, and D. F. M. Goodman. Graph Drawing by Stochastic Gradient Descent. *IEEE Transactions on Visualization and Computer Graphics*, 25(9):2738–2748, 2019. doi: 10.1109/TVCG.2018.2859997 2, 3, 4