

Hagrid – Gridify Scatterplots with Hilbert and Gosper Curves

Rene Cutura
rene.cutura@tuwien.ac.at
TU Wien & University of Stuttgart
Austria

Cristina Morariu
cristina@morariu.ro
University of Stuttgart
Germany

Zhanglin Cheng
zhanglin.cheng@gmail.com
Shenzen Institutes of Advanced
Technology
China

Yunhai Wang
cloudseawang@gmail.com
Shandong University
China

Daniel Weiskopf
Daniel.Weiskopf@visus.uni-
stuttgart.de
University of Stuttgart
Germany

Michael Sedlmair
Michael.Sedlmair@visus.uni-
stuttgart.de
University of Stuttgart
Germany



Figure 1: A detail view on a section of the t-SNE projection of the Art UK Paintings dataset. Gridified with HAGRIDGC.

ABSTRACT

A common enhancement of scatterplots represents points as small multiples, glyphs, or thumbnail images. As this encoding often results in overlaps, a general strategy is to alter the position of the data points, for instance, to a grid-like structure. Previous approaches rely on solving expensive optimization problems or on dividing the space that alter the global structure of the scatterplot. To find a good balance between efficiency and neighborhood and layout preservation, we propose HAGRID, a technique that uses space-filling curves (SFCs) to “gridify” a scatterplot without employing expensive collision detection and handling mechanisms. Using SFCs ensures that the points are plotted close to their original position, retaining approximately the same global structure. The resulting scatterplot is mapped onto a rectangular or hexagonal grid, using Hilbert and Gosper curves. We discuss and evaluate the theoretic runtime of our approach and quantitatively compare our approach to three state-of-the-art gridifying approaches, DGRID, Small multiples with gaps SMWG, and CorrelatedMultiples CMDS, in an evaluation comprising 339 scatterplots. Here, we compute several quality measures for neighborhood preservation together with an analysis of the actual runtimes. The main results show that, compared to the best other technique, HAGRID is faster by a factor of four, while achieving similar or even better quality of the

gridified layout. Due to its computational efficiency, our approach also allows novel applications of gridifying approaches in interactive settings, such as removing local overlap upon hovering over a scatterplot.

CCS CONCEPTS

• **Human-centered computing** → **Visualization techniques.**

KEYWORDS

Space-filling curve, Grid layout, Neighborhood-preserving.

ACM Reference Format:

Rene Cutura, Cristina Morariu, Zhanglin Cheng, Yunhai Wang, Daniel Weiskopf, and Michael Sedlmair. 2021. Hagrid – Gridify Scatterplots with Hilbert and Gosper Curves. In *The 14th International Symposium on Visual Information Communication and Interaction (VINCI '21), September 6–8, 2021, Potsdam, Germany*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3481549.3481569>

1 INTRODUCTION

Scatterplots are widely used representations for 2D data. The position of the dot is the main visual encoding and allows the user to perceive proximity or similarity between individual data points. Additional channels, such as color, shape, and size, can be used to show other properties of the respective data point.

A possible enhancement for scatterplots is replacing dots with meaningful glyphs or images that provide additional semantic information [43]. For instance, the individual points can represent the handwritten digits from the MNIST dataset [7]. In that case, their positions are the 2D projections resulting from applying dimensionality reduction to the high-dimensional dataset. For datasets that are not sourced from images, we may use glyphs, which may have various shapes and sizes, such as circular or hexagonal ones.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

VINCI '21, September 6–8, 2021, Potsdam, Germany

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8647-0/21/09...\$15.00

<https://doi.org/10.1145/3481549.3481569>

In these cases, occlusions stemming from overlapping points might impede the readability of the scatterplot [18]. A common approach to dealing with this issue is to detect collisions between image or glyph points and slightly jitter or move them around to reduce the overlap. Handling collisions is also necessary in graph drawing, wordles, and in gridifying maps.

Techniques formulated as optimization problems [23, 28] produce overlap-free layouts with high quality, but come with high runtimes, impeding interactive applications. Techniques generating space-filling results have better runtimes [9, 18], but can only preserve the shape and patterns of the scatterplot if adding dummy points, which also increases the runtime complexity for large grids. Our work is primarily motivated by dimensionality reduction processes in which users have to frequently change the parameterization of the algorithms to find good results. For such applications, a good tradeoff between fast runtimes and maintaining the original global structure is important.

To address these issues, we propose HAGRID¹, a technique that uses space-filling curves (SFCs) to “gridify” a scatterplot. SFCs are created by a starting pattern repeatedly replacing the vertices of the pattern with the same pattern, but rotated and flipped, so that the ends of the pattern connect to each other. This process creates a self-similar and self-avoiding curve. Most importantly, this recursion is bijective, i.e., a point in the area of the SFC can be mapped to a 1D index on the curve, and then decoded back to the 2D domain. The vertices of the SFCs correspond to the centers of the squared (HC) or hexagonal (GC) cells on the resulting grid, where the glyph or image point will be ultimately plotted. We use these properties to align the points of a scatterplot on a grid and to handle collisions, i.e., points mapped to the same SFC vertex or the same cell on the grid. We resolve collisions on the curve by moving the respective point left or right on the SFC (see Algorithm 2).

To evaluate HAGRID, we quantitatively compare it with the three related approaches. To this end, we use 339 scatterplots and compute different quality metrics on neighborhood preservation and layout similarity, as well as runtimes. Our results indicate that HAGRID is substantially faster while keeping similar or even better visual quality. In summary, we make the following main contributions:

- HAGRID, a technique for aligning 2D points on a grid, defined using space-filling curves.
- The results of a quantitative evaluation comparing HAGRID to DGRID, CorrelatedMultiples (CMDS), and Small Multiples with Gaps (SMWG).

Equipped with this faster approach, we provide two case studies in the supplemental material illustrating how it can be used in interactive applications and to visualize large datasets. Our implementations of HAGRID and all evaluation metrics used are available, both in Python (<https://github.com/kix2mix2/Hagrid>) and JavaScript (<https://github.com/saehm/hagrid>).

2 RELATED WORK

We review existing techniques for collision removal or reduction in scatterplots and similar visual encoding techniques. Closest to our work are approaches that seek to remove overlap of point representations by altering their position. The problem of point

positioning occurs across different visualization types. We use the visualization type to categorize them into the following groups:

Graphs: Node Overlap Removal. The idea of removing node overlap in graph drawing is similar to our approach. For example, MIOLA [16] arranges rectangular boxes in a way such that no overlap occurs anymore and the neighborhood of a box is preserved as much as possible. It uses a mixed integer quadratic optimization formulation, which can be solved by interfacing to optimization engines. To align all points on a grid, MIOLA requires additional constraints, further increasing the runtime. The algorithm gets measured and compared with VPSC [10], Prism [13], Voronoi [8], and RWordle-C [38], which have the same goal and use case. For comparison, they use various quality metrics: Euclidean distance, layout similarity, orthogonal ordering, size increase, and neighborhood preservation. The selected datasets are video snippet collections. The runtime success is based on solving their proposed formulation with Gurobi, the fastest optimization engine commercially available. Nachmanson et al. [30] builds a minimum spanning tree and grows edges between colliding nodes. They compare their technique GTree with Prism [13] by measuring the area of the result, edge length dissimilarity, and procrustean similarity [4]. Marcilio-Jr et al. [24] also evaluate techniques [10, 13, 17, 33, 38] of node overlap removal. All of those techniques were compared to at least one of the baselines implemented for this paper, and hence were not selected for our evaluation.

Space-filling Treemaps. NMAP [9] seeks to generate a space-filling treemap of a given visual area. Starting with a scatterplot, it recursively replaces the individual dots with unevenly sized rectangular boxes, which eventually form a treemap. They compare their method with OOT (One-dimensional Ordered Treemap) and SOT (Spatially-ordered Treemap) by comparing the aspect-ratio of individual boxes for each point, displacement, and neighborhood preservation. They evaluate the relationship between runtime and number of points on nine generated datasets. NMAP, OOT, and SOT have in common that areas should be transformed in such a way that the proximity between objects remains intact, thereby filling the whole visualization plane. NMAP has an extension that adds points in such a way that the algorithm results in a layout where all generated bounding-boxes have the same size. By comparison, our technique preserves as much as possible the distances (i.e., empty spaces) between points and generates equally sized squares or hexagons by default.

Maps. Several techniques exist for arranging geographical entities to uniform tiles, for instance, small multiples with gaps (SMWG [28]), generating tile maps [26], or coherent grid maps [29]. These methods work on the centroids of geographical areas, trying to maintain the global shape while keeping potential adjacency of respective areas. Evaluating the use case of these techniques needs additional metrics, for instance, orthogonal ordering is important. Simplifying maps has a high demand on visual quality to keep the map recognizable, but runtimes are relatively unimportant as they are mostly used in a static setting. In contrast, we focus on use cases like gridifying dimensionality reduction and interactive settings, in which runtime plays a central role.

¹HAGRID is short for Hilbert And Gosper Curve-based GRIDs

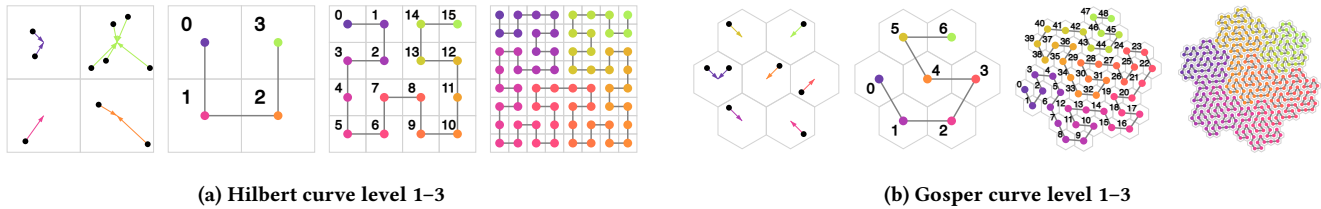


Figure 2: In (a) and (b) on the left, we show a set of randomly generated 2D points mapped to the Hilbert and Gosper curves at the first level of depth. In the other images of (a) and (b), we show the index number of each vertex for the two curves. In the transformation from 2D to 1D, each point is assigned its own index on the Hilbert/Gosper curve. In the level 1 depictions (left), points collide because there are not enough cells to fit all the points. As levels increase, collisions decrease but can still occur. Algorithm 2 adapts the initially assigned position to the closest available one on the curve. Each step produces m times the number of cells in the previous level. So, depending on the number of cells in the starting pattern, the SFC has m^l cells at the l th level (see Table 1).

Scatterplots: GRIDFIT [20] uses quad trees to gridify scatterplots and proposes two baselines for their evaluation: a naive approach using nearest neighbors to find the best available cell to place colliding point, and one employing SFCs to handle overlap. Their SFC baseline shares some commonalities to our method, but only uses the SFC for the grid coordinate computation, whereas HAGRID uses the SFCs for collision handling, which constitute the biggest runtime efficiency. GRIDFIT, as presented in the paper, is not entirely reproducible and none of the related work, that we are aware of, evaluated against it.

CorrelatedMultiples (CMDS) [23] use a variation of MDS to “gridify” data plots. Their use case is to show uniformly-sized small multiples instead of dots in a scatterplot to enrich the visualization with more information. They follow an approach similar to a force-directed layout method, usually employed for graphs. Their evaluation focuses on the runtime analysis. They compare their technique against SpatialGrid [46], and GridMap [11]. They also conduct a user study for analyzing the usefulness of small multiples in a scatterplot with favorable results.

DGRID [18] takes a similar space dividing approach to GRIDFIT, and bisects the visual space repeatedly, so that each point has its own rectangular or squared grid cell while preserving neighborhoods. They compare their technique with Kernelized Sorting [34], Self-Sorting Map [39], and IsoMatch [12]. This evaluation is the most extensive one, by evaluating neighborhood preservation, layout similarity, and cross-correlation on a range of datasets from the *UCI Machine Learning Repository*.

SMWG, CMDS and DGRID address the same use case as we do and are, thus, primary candidates for comparison. We have selected some of the quality metrics that they used in their individual evaluations (see Section 4).

3 TECHNIQUE

This section starts by providing some background on space-filling curves (SFCs), and specifically Hilbert and Gosper curves. We also derive a list of properties that are beneficial for our goal.

3.1 Background and Properties of Space-filling Curves

SFCs start with a pattern that is recursively repeated. For the Hilbert curve (HC) [19], it is the pattern $\sqcup\sqcup$, creating a grid of squares indexed continuously from 0 to m^l , where m is the number of vertices in the pattern and l the number of recursions, referred in this paper as the (depth) level of the curve. The Gosper curve (GC) [14] creates a hexagonal grid with the start-pattern $\begin{smallmatrix} \nearrow \\ \square \\ \searrow \end{smallmatrix}$. Examples of HC and GC at various levels are available in Figure 2.

Such techniques take advantage of the fast mapping either from 2D to 1D or vice versa. They use this mapping and the neighborhood-preservation property to visualize different aspects of specific data. Our technique does not focus on mapping data for a specific use case. Our primary goal is more general, i.e., creating a gridified layout without any overlap by re-mapping points. While we illustrate our approach with scatterplots, it can be used for any data that can be processed as a set of (x, y) coordinates, for instance, node-link diagrams or point clouds.

In our approach, we specifically use Hilbert and Gosper curves, as they have some desirable properties for visualization use cases that we seek to leverage with our approach:

- **Space-filling** – as the level of the curve increases toward infinity, every high-dimensional point can be represented by a vertex of the 1D curve. This ensures there is no constraint on either the dimensionality or the number of items to be visualized [41].
- **Bijectivity** – 2D points can be mapped to the 1D curve and back in the 2D plane, allowing us to switch between the two representations [44].
- **Neighborhood-preservation** – a point always retains approximately the same position on the curve regardless of whether the level of the curve is increased or decreased [44].
- **Stability** – small changes in the position on a curve yield the smallest possible change in the 2D outputs [41].

3.2 Proposed Technique

With HAGRID, our goal is to create overlap-free, gridified scatterplots where the dots are replaced with images or glyphs. Our technique differs from the space-filling approaches (DGRID, NMAP)

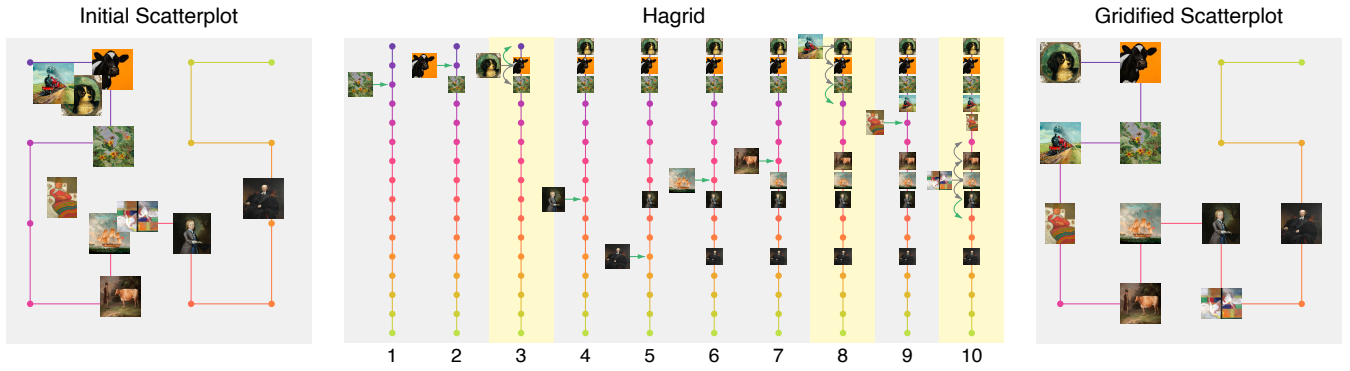


Figure 3: Illustration of Algorithm 1 and Algorithm 2 of HAGRID using the Hilbert curve. An initial scatterplot (left) consisting of 10 images of the *Art UK Paintings* dataset is gridified with Algorithm 1. One image after another gets assigned to a vertex of the Hilbert curve (middle). If a collision occurs (yellow background), Algorithm 2 resolves it by moving the image to another vertex. The first collision occurs at step 3 and gets resolved by moving it to the next empty spot to the left. The next collision happens in the same area at step 8. Algorithm 2 finds the next empty valid spot on the right, to the left are no spots available. The last collision appears at the end (step 10). Algorithm 2 finds two empty spot to the left and right. The collision gets resolved by putting the respective point to the right, because the 2D distance to the right spot is smaller.

above in that we do not only seek to preserve the local neighborhoods, but also the global structure of a scatterplot. By global structure, we mean the preservation of the characteristics of a set of points projected into 2D Euclidean space. These properties can include, but are not limited to, measures such as density, skewness, shape, and outliers [37, 45]. Loss of global structure implies that information like outliers or point density is lost in the resulting layout. CMDS and SMWG try to retain the global structure of the visualization, but have high runtimes.

HAGRID consists of following steps:

- (1) Begin by setting the level l of the used SFC, where $l \geq l_{\min}$ (see Equation 1).
- (2) Assign the datapoints to a grid defined by level l and the used SFC, $g_l : \mathbb{R}^2 \rightarrow G_l = \{(i, j)\}$, where the pairs (i, j) represent the possible coordinates of the grid G_l (see Figure 2).
- (3) Use $f_l : G_l \rightarrow I_l \subset \mathbb{N}$, where I_l is the set of indices or vertices of the SFC of level l .
- (4) If a point is assigned to an already occupied vertex of the SFC, Algorithm 2 resolves the collision.
- (5) Finally, $f_l^{-1} : I_l \rightarrow G_l$ maps the points back onto the grid G_l , resulting in an overlap-free scatterplot.

By doing so, HAGRID transforms the continuous 2D problem into a discretized 1D problem. This transformation allows us to positively impact the runtime of our algorithm.

The function g_l maps the original 2D coordinates to the SFC. First, the data is transformed to fit into the boundaries of grid G_l . Then, the transformed coordinates get rounded to the grid cell (i, j) that the datapoint should be assigned to. The main added value of using the SFC is, therefore, the collision handling.

Figure 3 conceptually illustrates the process introduced above, using images from the public *Art UK Paintings* dataset [6] instead of dots. The mapping $(f_l \circ g_l) : \mathbb{R}^2 \rightarrow I_l$ (steps 2 & 3 in the list) returns an index, transforming the position of the 2D point to a 1D vertex on the SFC (see Figure 3 middle). The SFC needs to be deep

enough to hold all the points of the used dataset (see Equation 1). We loop through each point in the list and assign it a 1D index on the SFC. Collision handling happens in the 1D space, on the fly. If a new point is assigned to an occupied vertex, we move the point to the left or to the right, based on which direction offers the closest empty spot (yellow areas in Figure 3). After the final point positioning, all collisions are resolved (Figure 3 right). Finally, the 1D vertices are transformed back to the 2D grid G_l , using function $f_l^{-1} : I_l \rightarrow G_l$. The entire process is detailed in Algorithm 1.

The final set of coordinates represents the centers of square or hexagonal cells of uniform sizes. The size of the cells is determined by the level of depth of the curve (l), introduced in Section 3.1. The depth level is, therefore, a parameter of our technique. Since our goals cover both being able to plot images or glyphs, and completely avoiding overlaps, we can reformulate them as follows: we want to plot a set of images as large as possible without overlap.



Given the space-filling property of the curves, if we were to generate deep enough curves, no overlap would happen, as the curve would cross through all 2D space as $l \rightarrow \infty$. In other words, the gridified structure would approach the original layout of the scatterplots. However, a higher curve level l also leads to a more fine-grained grid, and hence smaller cells where the glyphs and images can be plotted. For this reason, we want to calculate the minimal level of depth according to the number of points to be plotted. This l_{\min} is the default value and calculated by:

$$l_{\min} = \lceil \log_m n \rceil, \quad (1)$$

where n is the number of points to be scattered, and m is the number of vertices in the initial recursion pattern (i.e., the SFCs at level 1). For HC (\downarrow), $m = 4$ and for GC (\swarrow), $m = 7$ (see Figure 2). The users may always alter the level to achieve different results depending on their individual goals. For example, if analyzing the global structure of a scatterplot is more important, we recommend setting the level to $l_{\min} + 1$ or higher, or using the level of depth that guarantees a user-defined minimum percentage of whitespace, by using $n \cdot (1+w)$

in Equation 1 instead of n , where w is the percentage of whitespace. For example, the level with guaranteed 50% whitespace is calculated with $l_{50\%} = \lceil \log_m(n \cdot 1.5) \rceil$. Table 1 lists the maximum number of cells available for different levels.

Table 1: The maximum number of points that can be encoded in an SFC of a particular level.

l_{\min}	1	2	3	4	5	6	7
	4	16	64	256	1024	4096	16384
	7	49	343	2401	16807	117649	823543

The process of placing a point onto the SFC is similar to inserting a value into a hash table. The recursion depth l of the SFC depends on the number of points in the scatterplot (see Equation 1), and defines the required number of steps of the index computation of a single point. The computation of an index assignment of one point using $(f_l \circ g_l) : \mathbb{R}^2 \rightarrow I_l$ needs $\mathcal{O}(l)$ time, or — as l depends on n (Equation 1) — $\mathcal{O}(\log n)$ time. For all points, the overall runtime is thus $\mathcal{O}(n \cdot l) = \mathcal{O}(n \log n)$.

Algorithm 1: Gridify scatterplot.

Data: $X = \{x_1, x_2, \dots, x_n\}$, where X is the set of the datapoints that should get gridified
Input: The dataset X , the mapping functions g_l, f_l and f_l^{-1} , the depth of the space-filling curve $l \geq l_{\min}$.
Result: $Y = \{y_1, y_2, \dots, y_n\}$, the set of datapoints layed out on the respective grid.
 $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n\}$, where $\tilde{x}_i = g_l(x_i), \forall i \in \{1, \dots, n\}$
 $P = \text{new Map}()$
foreach $i \in \{1, \dots, n\}$ **do**
 $p_i = f_l(\tilde{x}_i)$
 // check for collision.
 if $P.has(p_i)$ **then**
 $p_i = \text{solveCollision}(p_i, x_i, f_l^{-1})$
 $P.set(p_i, i)$
// remap to 2D coordinates.
 $Y = \text{new Array}(n)$
foreach $i \in \{1, \dots, n\}$ **do**
 $y_i = f_l^{-1}(p_i)$
return Y

The above considerations are valid as long as there is no collision on the SFC. This best case occurs if the points are evenly distributed in the sense that there is at the most one point per index on the SFC from the mapping by f . However, collisions will happen in most cases. Additional computational costs can be introduced by collision handling. The worst case happens if all points are initially mapped by f to the same SFC index. In this case, collision handling will take as many steps as there are points already placed on the SFC. When placing n points subsequently, this will lead to an average of $n/2$ collision-handling steps per point, or $\mathcal{O}(n^2)$ steps to handle all n points. In total, the runtime will therefore be $\mathcal{O}(n \log n + n^2) = \mathcal{O}(n^2)$. The first term $\mathcal{O}(n \log n)$ comes from the initial placement of the points and the second term $\mathcal{O}(n^2)$ deals with collision handling. Therefore, the worst case runtime is governed by collision handling.

Algorithm 2: Solve collision.

function solveCollision(p, x_i, f_l^{-1})
 $p_{left} = p - 1$
 $p_{right} = p + 1$
 while No empty valid spot found **do**
 $p_{left} = p_{left} - 1$
 $p_{right} = p_{right} + 1$

 // Empty spot found!
 // Return closest valid SFC index.
 return closestAndValid $_{x_i, f_l^{-1}}(p_{left}, p_{right})$

However, both extreme cases are not typical for our applications. For in-between cases, it is critical to model the number of collision-handling steps, depending on n . Let us assume that the function $c(n)$ provides the average number of collision-handling steps. Then, we arrive at a total runtime of $\mathcal{O}(n \log n + c(n) \cdot n)$, composed of the initial placement of points and the subsequent handling of collisions for all n points. Of course, $c(n)$ depends heavily on the distribution of points in 2D and, therefore, cannot be discussed here in full detail. We refer to generative data models [36] for some approaches to model data with certain characteristics that could be related to the properties relevant for collision handling. Even without such a data model, we know that the number of collision handling steps are bounded: $0 \leq c(n) \leq n$, comprising the best and worst case scenarios from above. While $c(n)$ is a theoretical model, Figure 5 shows the number of actual collisions for randomly generated uniformly distributed datasets, as well as their impact on runtime. Therefore, our runtime will then be between $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$.

4 QUANTITATIVE ANALYSIS

In this section, we present a quantitative evaluation of HAGRID in comparison to selected techniques.

4.1 Competing Techniques

We compare our technique with its two versions, Hilbert curve (HAGRID_{HC}) and Gosper curve (HAGRID_{GC}), to three state-of-the-art approaches: CMDS, DGRID, and SMWG. We selected these candidates based on two criteria: (i) how well they address our use cases, and (ii) whether the techniques had not been previously compared against each other. In previous evaluations against related methods [11, 12, 39, 46], these approaches showed very convincing results in terms of runtime and quality metrics.

All in all, the selected methods represent some of the best available techniques that address the problem of point positioning, and more specifically, our scatterplot layout use case. A side contribution of this paper are the Python and JavaScript implementations of the used techniques (except SMWG), as well as of the evaluation metrics that will be presented in the next section.

4.2 Evaluation Metrics

For the comparison, we have selected the following evaluation metrics: neighborhood preservation (NP , also called $AUC_{\log RNX}$ [22]),

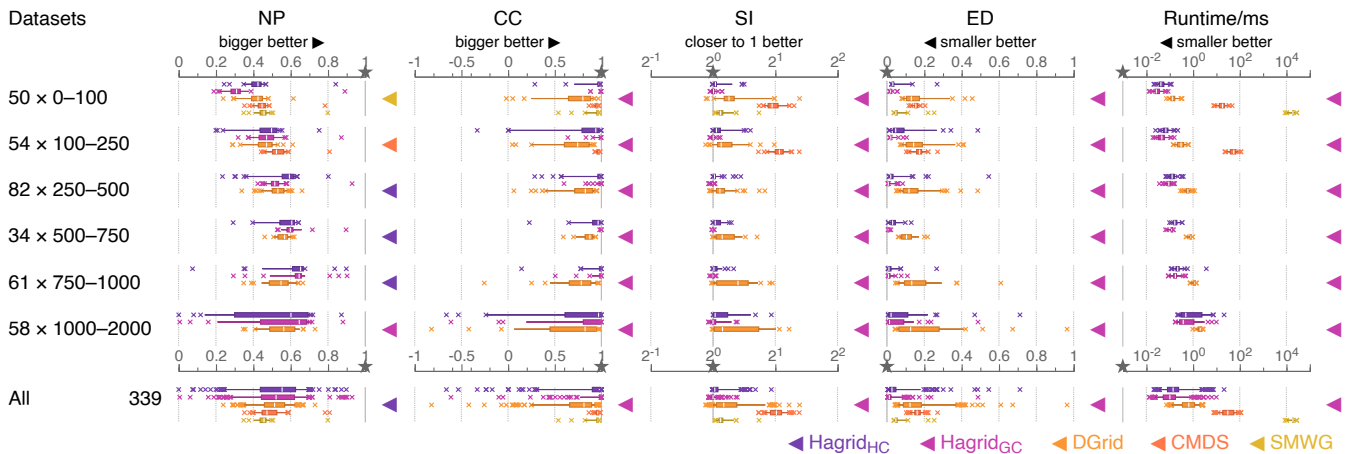


Figure 4: Results of our evaluation of the 339 scatterplots, aggregated by dataset size, and with all data in the last row. The boxplots show the median, the 25% and 75% quantiles, while the ends show the 5- and the 95 percentile. The color of the triangles indicates the best method, measured by the median of the particular metric. The results demonstrate that in each metric HAGRID performs better than the other techniques, except for NP for dataset sizes smaller than 250 points. The runtime (log₁₀-scale) of HAGRID is better than any other technique. The fastest alternative is DGRID, which is four times slower though. The outliers of HAGRID_{HC} and HAGRID_{GC} for NP in the last group stem from some projections of the *paris buildings* dataset, see the supplemental materials for more detail.

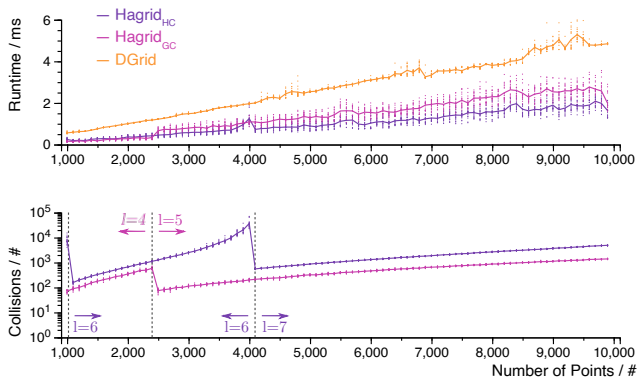


Figure 5: Charts showing runtime evaluation of generated data (uniformly distributed points, with values from $[0, 1]$ for each dimension) with specific dataset sizes (x -axis) for HAGRID_{HC}, HAGRID_{GC}, and DGRID. We computed the grid layouts with the three techniques for dataset sizes from 1,000 to 10,000 with a step size of 100, 20 times for each dataset size and each method. The lines show the connected means per dataset size. We measured the runtime (upper chart in ms) and the number of collisions occurred (bottom chart, log-scale). The charts show that — as expected — more collisions increase the runtimes slightly (for HAGRID_{HC} at $n = 4,000$ and for HAGRID_{GC} at $n = 2,400$).

cross-correlation (CC), Euclidean distance (ED), size increase (SI), and runtime (RT). The metrics in this list were selected based on the ones mentioned in the related work. Details of the metrics can be found in the supplemental materials.

Our motivation behind selecting these particular metrics is two-fold. Neighborhood preservation and cross-correlation describe

whether the local neighborhoods are preserved. The other metrics, Euclidean-distance and size increase, are more sensitive to how the global aspect of the scatterplot changes. We have selected those metrics, because our main goal is to remove overlap, while roughly holding the position of the points in the original scatterplots. The local metrics neighborhood preservation and cross-correlation will help us compare against DGRID where the overarching goal was space-filling while maintaining neighborhoods. To make the metrics comparable, we scaled the extents of all original and gridified scatterplots to $[0, 1]$.

4.3 Evaluation Data

For the evaluation, we gathered 60 real and synthetic datasets. 54 of these datasets stem from Aupetit and Sedlmair [1]. We have further included six additional image datasets publicly available online. From these datasets, we created a total of 339 *scatterplots* by projecting them with different dimensionality reduction (DR) methods. These methods lead to vastly diverse layouts with different distributions of points across them. Therefore, they allow us to assess how well HAGRID_{HC}, HAGRID_{GC}, and the other techniques perform depending on the number of points in the dataset, and on the number of total collisions.

In the supplemental material, we provide a list of the datasets involved in the evaluation. We include their names, sizes, and number of projections computed on each. In terms of DR algorithms, we used PCA [32], robust PCA [5], t-SNE [42], Isomap [40], LLE [35], UMAP [25], Spectral Embedding [2], Gaussian Random Project [3], and MDS [21]. For parametric DRs, we performed a grid search through a range of values for each parameter involved. The final 339 scatterplots were selected by uniformly sampling across different scatterplot sizes from a total of 1695 projections.

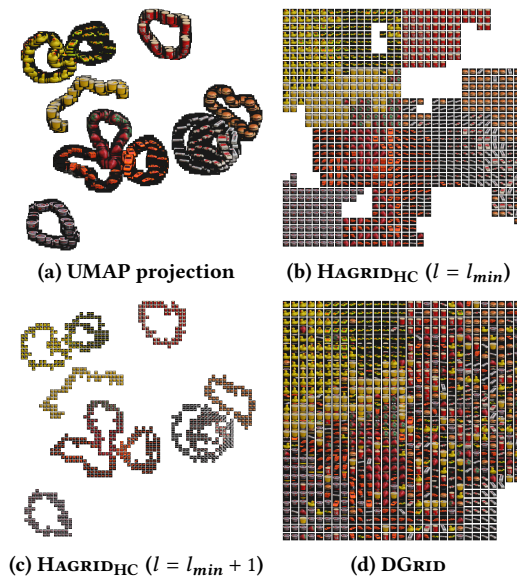


Figure 6: The UMAP projection of the *COIL-100* dataset with overlaps, and gridified versions of the projection.

4.4 Evaluation Setup

We implemented HAGRID_{HC} and HAGRID_{GC}, as well as CMDS and DGRID in Python and JavaScript. For SMWG, we used the available online tool [27] as we could not replicate the code based on publicly available information. All evaluations were run on an Intel(R) Core(TM) i7-8705G CPU @ 3.10 GHz, with 16 GB of RAM.

We categorized the 339 scatterplots into six groups according to the number of points (see Figure 4). HAGRID_{HC}, HAGRID_{GC}, and DGRID were computed for all 339 scatterplots. CMDS were only computed for scatterplots up to 250 points (group 1 and 2), and SMWG only for scatterplots up to 100 points (group 1). The reason behind this choice is that the computation times of these two techniques became intractable for larger datasets. These computation times were far beyond the requirements of our use case for rapid or even interactive application of such approaches. Last, we computed the quality metrics for all results.

The supplemental material includes further details and the code to reproduce the evaluation results.

4.5 Results

Figure 4 summarizes the results of our comparative evaluation according to the four quality metrics introduced and the runtime. We investigate the scalability of HAGRID, by aggregating the results according to dataset size, and an extra evaluation only measuring the runtime in Figure 5. The supplemental material contains a more detailed version of Figure 4, which shows the results for each dataset separately. As we fixed CMDS’ number of maximum rounds to 50, not all overlap was removed from the final layout, and the quality metrics should be slightly more optimistic as the 50th round layout will always be closer to the original.

Generally, HAGRID’s runtime outperforms every other method we evaluated against and is about four times as fast compared to the next best technique, DGRID, across different dataset sizes. HAGRID

outperforms the other methods in almost all metrics, except for NP. SMWG and CMDS produce generally good quality results, but at the cost of much higher runtimes. In group 1 (0 – 100 points), HAGRID needs roughly a millisecond to gridify the result, whereas SMWG needed on average 30 seconds. Please note that, in Figure 4, we use a \log_{10} -scale for runtime and a \log_2 -scale for the SI metric to improve readability.

Finally, we show in Figure 6 the resulting grids for the UMAP-reduced *COIL-100* dataset [31]. We compare the original plot in Figure 6a, the gridified results for HAGRID, computed for both minimum available level (in Figure 6b) and a higher depth level (in 6c), and the DGRID version (6d). The runtime for CMDS was too high, and the online-tool for SMWG even crashed for this dataset, which has 792 samples. Further qualitative results and examples are available in the supplemental materials.

5 CONCLUSIONS AND FUTURE WORK

A limitation of HAGRID and other gridifying techniques occurs in scatterplots with very dense or differently dense regions. To maintain the global structure of differently dense or very dense areas of scatterplots with HAGRID, high levels of the respective SFC are required, which can lead to very small grid cells. We found this behavior also in our analysis, where some of the scatterplots led to bad results, especially for the NP metric (see Figure 4). HAGRID is sensitive to extreme outliers, which entail non-outlying points being squished into very small areas in the scatterplot. Such situations are sub-optimal for filling a static SFC grid as many collisions occur (see supplemental material for examples).

An additional limitation is inherited from the SFCs used, Hilbert and Gosper curves, which are not circular. When collisions happen close to the ends of the curves, Algorithm 2 has an imbalanced number of vertices to the left and to the right, which might worsen the quality of the result. In the future, we want to replace the SFCs used with their circular versions. The Moore curve [15] is, for instance, the circular alternative of the Hilbert curve. We used the Hilbert curve and the Gosper curve because these implementations have better computational efficiency.

In this paper, we proposed HAGRID, a novel approach for generating gridified layouts that preserves the local and global neighborhoods and cluster structure of a scatterplot, while removing the overlap of point representations. We demonstrated the approach by employing two space-filling curves, Hilbert and Gosper, to create squared and hexagonal grids. The set of comparisons we provide shows that our technique outperforms the existing state-of-the-art techniques in terms of different quality metrics, and is roughly four times faster than them.

ACKNOWLEDGMENTS

This work was supported by the BMK FFG ICT of the Future program via the ViSciPub project (no. 867378), and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 251654672 – TRR 161.

REFERENCES

- [1] Michael Aupetit and Michael Sedlmair. 2016. SepMe: 2002 New visual separation measures. In *IEEE Pacific Vis. Symp. (PacificVis)*. IEEE, 1–8. <https://doi.org/10.1109/PACIFICVIS.2016.7465244>

- [2] Mikhail Belkin and Partha Niyogi. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation* 15, 6 (2003), 1373–1396. <https://doi.org/10.1162/08997660327180317>
- [3] Ella Bingham and Heikki Mannila. 2001. Random projection in dimensionality reduction: Applications to image and text data. In *Proc. ACM Intl. Conf. Knowledge Discovery and Data Mining (SIGKDD)*. 245–250. <https://doi.org/10.1145/502512.502546>
- [4] Ingwer Borg and Patrick Groenen. 2003. Modern Multidimensional Scaling: Theory and Applications. *J. Educational Measurement (JEM)* 40, 3 (2003), 277–280. <https://doi.org/10.1111/j.1745-3984.2003.tb01108.x>
- [5] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. 2011. Robust Principal Component Analysis? *J. ACM* 58, 3, Article 11 (June 2011), 37 pages. <https://doi.org/10.1145/1970392.1970395>
- [6] Elliot Crowley and Andrew Zisserman. 2014. The State of the Art: Object Retrieval in Paintings using Discriminative Regions. In *Proc. British Machine Vision Conf.* BMVA Press.
- [7] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142. <https://doi.org/10.1109/MSP.2012.2211477>
- [8] Qiang Du, Vance Faber, and Max Gunzburger. 1999. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Rev.* 41, 4 (1999), 637–676. <https://doi.org/10.1137/S0036144599352836>
- [9] Felipe SLG Duarte, Fabio Sikansi, Francisco M Fatore, Samuel G Fadel, and Fernando V Paulovich. 2014. Nmap: A Novel Neighborhood Preservation Space-filling Algorithm. *IEEE Trans. Visualization & Computer Graphics (TVCG)* 20, 12 (2014), 2063–2071. <https://doi.org/10.1109/TVCG.2014.2346276>
- [10] Tim Dwyer, Kim Marriott, and Peter J Stuckey. 2006. Fast Node Overlap Removal—Correction. In *Int. Symp. Graph Drawing*. Springer, 446–447. https://doi.org/10.1007/978-3-540-70904-6_44
- [11] David Eppstein, Marc van Kreveld, Bettina Speckmann, and Frank Staals. 2015. Improved Grid Map Layout by Point Set Matching. *Int. J. Comp. Geometry & Applications* 25, 02 (2015), 101–122. <https://doi.org/10.1142/S0218195915500077>
- [12] Ohad Fried, Stephen DiVerdi, Maciej Halber, Elena Sizikova, and Adam Finkelstein. 2015. IsoMatch: Creating Informative Grid Layouts. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 155–166. <https://doi.org/10.1111/cgf.12549>
- [13] Emden R Gansner and Yifan Hu. 2008. Efficient Node Overlap Removal Using a Proximity Stress Model. In *Int. Symp. Graph Drawing*. Springer, 206–217. https://doi.org/10.1007/978-3-642-00219-9_20
- [14] Martin Gardner. 1976. Mathematical games—in which “monster” curves force redefinition of the word “curve”. *Scientific American* 235, 6 (1976), 124–133.
- [15] Rowdra Ghatak, Manimala Pal, Chiranjibi Goswami, and DR Poddar. 2013. Moore Curve Fractal-Shaped Miniaturized Complementary Spiral Resonator. *Microwave and Optical Technology Letters* 55, 8 (2013), 1950–1954. <https://doi.org/10.1002/mop.27682>
- [16] Erick Gomez-Nieto, Wallace Casaca, Luis Gustavo Nonato, and Gabriel Taubin. 2013. Mixed Integer Optimization for Layout Arrangement. In *Symp. Graphics, Patterns and Images (SIBGRAPI)*. IEEE, 115–122. <https://doi.org/10.1109/SIBGRAPI.2013.25>
- [17] Erick Gomez-Nieto, Frizzi San Roman, Paulo Pagliosa, Wallace Casaca, Elias S Helou, Maria Cristina F de Oliveira, and Luis Gustavo Nonato. 2013. Similarity Preserving Snippet-Based Visualization of Web Search Results. *IEEE Trans. Visualization & Computer Graphics (TVCG)* 20, 3 (2013), 457–470. <https://doi.org/10.1109/TVCG.2013.242>
- [18] Gladys Hilarasaca and Fernando V Paulovich. 2019. Distance Preserving Grid Layouts. *arXiv preprint* (2019). <http://arxiv.org/abs/1903.06262>
- [19] David Hilbert. 1935. Über die stetige Abbildung einer Linie auf ein Flächenstück. In *Dritter Band: Analysis: Grundlagen der Mathematik- Physik Verschiedenes*. Springer, 1–2. https://doi.org/10.1007/978-3-662-38452-7_1
- [20] D. A. Keim and A. Herrmann. 1998. The Gridfit algorithm: an efficient and effective approach to visualizing large amounts of spatial data. In *Proc. Visualization*. 181–188. <https://doi.org/10.1109/VISUAL.1998.745301>
- [21] Joseph B Kruskal. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29, 1 (1964), 1–27. <https://doi.org/10.1007/BF02289565>
- [22] John A Lee, Diego H Peluffo-Ordóñez, and Michel Verleysen. 2015. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing* 169 (2015), 246–261. <https://doi.org/10.1016/j.neucom.2014.12.095>
- [23] Xiaotong Liu, Yifan Hu, Stephen North, and Han-Wei Shen. 2018. Correlated Multiples: Spatially Coherent Small Multiples With Constrained Multi-Dimensional Scaling. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 7–18. <https://doi.org/10.1111/cgf.12526>
- [24] Wilson E Marcílio-Jr, Danilo M Eler, Rogério E Garcia, and Ives R Venturini Pola. 2019. Evaluation of approaches proposed to avoid overlap of markers in visualizations based on multidimensional projection techniques. *Information Visualization* 18, 4 (2019), 426–438. <https://doi.org/10.1177/1473871619845093>
- [25] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426* (2018). <https://arxiv.org/abs/1802.03426>
- [26] Graham McNeill and Scott A Hale. 2017. Generating tile maps. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 435–445. <https://doi.org/10.1111/cgf.13200>
- [27] Wouter Meulemans, Jason Dykes, Aidan Slingsby, Cagatay Turkey, and Jo Wood. [n. d.]. Small Multiples with Gaps. <http://www.gicentre.org/smwg/>
- [28] Wouter Meulemans, Jason Dykes, Aidan Slingsby, Cagatay Turkey, and Jo Wood. 2016. Small Multiples with Gaps. *IEEE Trans. Visualization & Computer Graphics (TVCG)* 23, 1 (2016), 381–390. <https://doi.org/10.1109/TVCG.2016.2598542>
- [29] Wouter Meulemans, Max Sondag, and Bettina Speckmann. 2020. A Simple Pipeline for Coherent Grid Maps. *IEEE Trans. Visualization & Computer Graphics (TVCG)* (2020). <https://doi.org/10.1109/TVCG.2020.3028953>
- [30] Lev Nachmanson, Arlind Nocaj, Sergey Bereg, Leishi Zhang, and Alexander Holroyd. 2016. Node Overlap Removal by Growing a Tree. In *Int. Symp. Graph Drawing and Net. Vis.* Springer, 33–43. https://doi.org/10.1007/978-3-319-50106-2_3
- [31] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. 1996. Columbia object image library (coil-20). (1996).
- [32] Karl Pearson. 1901. LIII. On Lines and Planes of Closest Fit to Systems of Points in Space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572. <https://doi.org/10.1080/14786440109462720>
- [33] Roberto Pinho, Maria Cristina F de Oliveira, and Alneu de A. Lopes. 2009. Incremental board: a grid-based space for visualizing dynamic data sets. In *ACM Symp. Applied Computing (SAC)*. 1757–1764. <https://doi.org/10.1145/1529282.1529679>
- [34] Novi Quadrianto, Alexander J Smola, Le Song, and Tinne Tuytelaars. 2010. Kernelized Sorting. *IEEE Trans. Pattern Analysis and Machine Intelligence* 32, 10 (2010), 1809–1821. <https://doi.org/10.1109/TPAMI.2009.184>
- [35] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290, 5500 (2000), 2323–2326. <https://doi.org/10.1126/science.290.5500.2323>
- [36] Christoph Schulz, Arlind Nocaj, Mennatallah El-Assady, Steffen Frey, Marcel Hlawatsch, Michael Hund, Grzegorz Karch, Rudolf Netzel, Christin Schätzle, Miriam Butt, Daniel A. Keim, Thomas Ertl, Ulrik Brandes, and Daniel Weiskopf. 2016. Generative Data Models for Validation and Evaluation of Visualization Techniques. In *Proc. Beyond Time and Errors: Novel Evaluation Methods for Visualization (BELIV)*. 112–124. <https://doi.org/10.1145/2993901.2993907>
- [37] Michael Sedlmair, Andrada Tatu, Tamara Munzner, and Melanie Tory. 2012. A Taxonomy of Visual Cluster Separation Factors. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1335–1344. <https://doi.org/10.1111/j.1467-8659.2012.03125.x>
- [38] Hendrik Strobelt, Marc Spicker, Andreas Stoffel, Daniel Keim, and Oliver Deussen. 2012. Rolled-out Wordles: A Heuristic Method for Overlap Removal of 2D Data Representatives. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1135–1144. <https://doi.org/10.1111/j.1467-8659.2012.03106.x>
- [39] Grant Strong and Minglun Gong. 2014. Self-Sorting Map: An Efficient Algorithm for Presenting Multimedia Data in Structured Layouts. *IEEE Trans. Multimedia* 16, 4 (2014), 1045–1058. <https://doi.org/10.1109/TMM.2014.2306183>
- [40] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290, 5500 (2000), 2319–2323. <https://doi.org/10.1126/science.290.5500.2319>
- [41] Vojtěch Uher, Petr Gajdoš, Václav Snášel, Yu-Chi Lai, and Michal Radecký. 2019. Hierarchical Hexagonal Clustering and Indexing. *Symmetry* 11, 6 (2019), 731. <https://doi.org/10.3390/sym11060731>
- [42] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. 9, Nov (2008), 2579–2605. <http://jmlr.org/papers/v9/vandermaaten08a.html>
- [43] Matthew O Ward. 2002. A Taxonomy of Glyph Placement Strategies for Multidimensional Data Visualization. *Information Visualization* 1, 3-4 (2002), 194–210. <https://doi.org/10.1057/PALGRAVE.IVS.9500025>
- [44] Martin Wattenberg. 2005. A note on space-filling visualizations and space-filling curves. In *Proceedings of the IEEE Information Visualization Symposium*. IEEE, 181–186. <https://doi.org/10.1109/INFVIS.2005.1532145>
- [45] Leland Wilkinson, Anushka Anand, and Robert Grossman. 2005. Graph-Theoretic Scagnostics. In *Proceedings of the IEEE Information Visualization Symposium*. IEEE, 157–164. <https://doi.org/10.1109/INFVIS.2005.1532142>
- [46] Jo Wood and Jason Dykes. 2008. Spatially Ordered Treemaps. *IEEE Trans. Visualization & Computer Graphics (TVCG)* 14, 6 (2008), 1348–1355. <https://doi.org/10.1109/TVCG.2008.165>