



REGULAR PAPER

Rene Cutura · Cristina Morariu · Zhanglin Cheng · Yunhai Wang · Daniel Weiskopf · Michael Sedlmair

Hagrid: using Hilbert and Gosper curves to gridify scatterplots

Received: 1 February 2022 / Revised: 3 April 2022 / Accepted: 5 May 2022 / Published online: 11 July 2022
© The Author(s) 2022

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.

Keywords Space-filling curve · Grid layout · Neighborhood-preserving

R. Cutura (✉) · C. Morariu · D. Weiskopf · M. Sedlmair
University of Stuttgart, Stuttgart, Germany
E-mail: Rene.Cutura@visus.uni-stuttgart.de

C. Morariu
E-mail: cristina@morariu.ro

D. Weiskopf
E-mail: Daniel.Weiskopf@visus.uni-stuttgart.de

M. Sedlmair
E-mail: Michael.Sedlmair@visus.uni-stuttgart.de

Z. Cheng
Shenzhen Institutes of Advanced Technology, Shenzhen, China
E-mail: zhanglin.cheng@gmail.com

Y. Wang
Shandong University Qingdao Campus, Qingdao, China
E-mail: cloudseawang@gmail.com

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 (Ward 2002). For instance, the individual points can represent the hand-written digits from the MNIST dataset (Deng 2012). In that case, their positions are the 2D projections resulting from applying dimensionality reduction to the high-dimensional dataset. Rather than using dots, color, or shape to represent the points, we can plot the small image of the hand-written digit, allowing the user gain more insight from the data.

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. In these cases, occlusions stemming from overlapping points might impede the readability of the scatterplot (Hilasaca and Paulovich 2019). 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 (Meulemans et al. 2016; Liu et al. 2018) produce overlap-free layouts with high quality, but come with high runtimes, impeding interactive applications. Techniques generating space-filling results have better runtimes (Hilasaca and Paulovich 2019; Duarte et al. 2014), 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, where the limit of the recursion leads to a space-filling curve (see Fig. 2).

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 (see Fig. 1 as example for the use of GC). 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 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>). This article is an extension of a previous conference paper on the same topic (Cutura et al. 2021).

^{Par8} HAGRID is short for Hilbert And Gosper Curve-based GRIDs



Fig. 1 A detail view on a section of the t-SNE projection of the *Art UK Paintings* dataset. Gridified with Hagrid_{GC}

2 Related work

Space-filling curves can be used in many different ways (Bader 2012), for instance, to lay out data in the cache, which improves the runtime of certain algorithms. Here, we provide related work on how space-filling curves are currently used in visualization use cases. Then, we review existing techniques for collision removal or reduction in scatterplots and similar visual encoding techniques.

2.1 Applications of space-filling curves

This section discusses existing techniques that use SFCs for visualization-related use cases.

Jigsaw (Wattenberg 2005) uses SFCs, such as Hilbert-curves, to generate a space-filling map, competing with treemap generation algorithms. Their SFC approach creates non-rectangular maps, which have several superior properties, such as better preservation of ordering of their tree visualization.

Buchmüller et al. (2018) use SFCs to create 2D visualizations of simulation processes, so-called MotionRugs. Space-filling curves allow them to transfer the position of agents in the simulation to a vertex on the curve. The curve then gets visualized as a one-pixel wide column. The artifacts from this process allow tracing of behaviors of the agents in the simulation.

Muelder and Ma (2008) use SFCs for node-link graph layouts. They compute a matrix ordering of the graph, and project this ordering on a space-filling curve, in this case a HC or a GC. Taking advantage of the neighborhood preservation property of SFCs, this leads to a graph layout where clustered nodes are close in the final visualization.

Gospermap (Auber et al. 2013) creates treemaps with non-rectangular shapes of hierarchical data, by laying it out on the Gosper Curve. Other works use space-filling curves for neighborhood search in point clouds (Jeżowicz et al. 2014).

These 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 the specific data. Our technique does not focus on mapping data for a specific use case. Our primary goal is more general, that is, 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.

2.2 Point positioning

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:

2.2.1 Graphs—Node overlap removal

The idea of removing node overlap in graph drawing is similar to our approach. For example, MIOLA (Gomez-Nieto et al. 2013a) 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 (Dwyer et al. 2006), Prism (Gansner and Hu 2008), Voronoi (Du et al. 1999), and RWordle-C Strobelt et al. (2012), 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. (2016) builds a minimum spanning tree and grows edges between colliding nodes. They compare their technique GTree with Prism (Gansner and Hu 2008) by measuring the area of the result, edge length dissimilarity, and procrustean similarity (Borg and Groenen 2003). Marcílio-Jr et al. (2019) also evaluate techniques (Pinho et al. 2009; Dwyer et al. 2006; Gomez-Nieto et al. 2013b; Strobelt et al. 2012; Gansner and Hu 2008) 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.

2.2.2 Space-filling Treemaps

NMAP (Duarte et al. 2014) 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.

2.2.3 Maps

Several techniques exist for arranging geographical entities to uniform tiles, for instance, small multiples with gaps (SMWG (Meulemans et al. 2016)), generating tile maps (McNeill and Hale 2017), or coherent grid maps (Meulemans et al. 2020). 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.

2.2.4 Scatterplots

DUST & MAGNET (D & M) (Vollmer and Döllner 2020) is a technique for creating overlap-free scatterplots. Virtual magnets attract or reject points depending on a given feature. It avoids overlaps with a continuous optimization process, by first checking from the target position of a point in a defined grid in different directions for an empty spot. After finding an empty spot, in a second step, each placed point looks “back” for all adjacent grid cells closer to the target point where the point should have been placed, and taking the nearer available space. This approach can be used if the input is high-dimensional, our technique gridifies already projected data.

GRIDFIT (Keim and Herrmann 1998) 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) (Liu et al. 2018) 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 (Wood and Dykes 2008), and GridMap (Eppstein et al. 2015). They also conduct a user study for analyzing the usefulness of small multiples in a scatterplot with favorable results.

DGRID (Hilasaca and Paulovich 2019) 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 (Quadrianto et al. 2010), Self-Sorting Map (Strong and Gong 2014), and IsoMatch (Fried et al. 2015). 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 Sect. 4). The metrics employed and the results of this comparison are presented in Sect. 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) (Hilbert 1935), it is the pattern

$$\begin{array}{c} \text{L} \\ \text{L} \end{array}$$

, 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) (Gardner 1976) creates a hexagonal grid with the start-pattern

$$\begin{array}{c} \text{V} \\ \text{V} \end{array}$$

. Examples of HC and GC at various levels are available in Fig. 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 (Uher et al. 2019).
- *Bijection* 2D points can be mapped to the 1D curve and back in the 2D plane, allowing us to switch between the two representations (Wattenberg 2005).
- *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 (Wattenberg 2005).

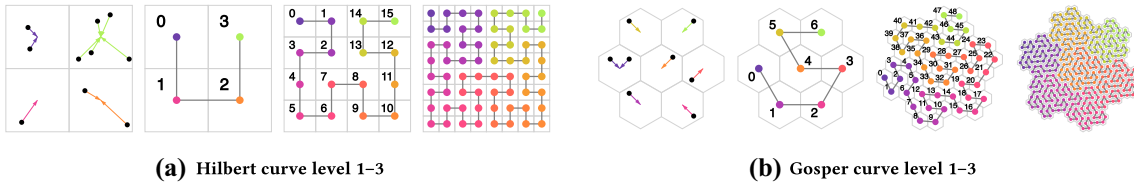


Fig. 2 In **a** and **b** on the left, we show a set of randomly generated 2D points mapped to the Hilbert and Gosper curves at the first level of depth. In the other images of **(a)** and **(b)**, we show the index number of each vertex for the two curves. In the transformation from 2D to 1D, each point 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)

- *Stability* Small changes in the position on a curve yield the smallest possible change in the 2D outputs (Uher et al. 2019).

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) 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 (Sedlmair et al. 2012; Wilkinson et al. 2005). 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 $l \geq l_{\min} = \lceil \log_m n \rceil$ (see Eq. 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 Fig. 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.

The function g_l maps the original 2D coordinates to the Grid G_l . 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 (Crowley and Zisserman 2014) 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 Fig. 3 middle). The SFC needs to be deep enough to hold all the points of the used dataset (see Eq. 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 Fig. 3). After the final point positioning, all collisions are resolved (Fig. 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 Sect. 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 (

□

), $m = 4$ and for GC (

✂

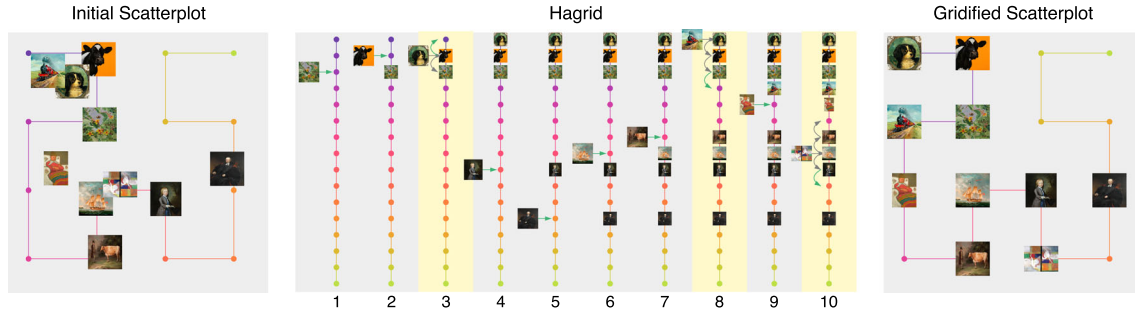


Fig. 3 Illustration of Algorithm 1 and Algorithm 2 of HAGRID using the Hilbert curve. An initial scatterplot (left) consisting of 10 images of the *Art UK Paintings* dataset is gridified with Algorithm 1. One image after another gets assigned to a vertex of the Hilbert curve (middle). If a collision 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

), $m = 7$ (see Fig. 2). The users may always alter the level to achieve different results depending on

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 $P.\text{has}(p_i)$ **then**
 $p_i = \text{solveCollision}(p_i, x_i, f_l^{-1})$
 $P.\text{set}(p_i, i)$
 // 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

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 Eq. 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
\sqcup	4	16	64	256	1024	4096	16,384
\boxplus	7	49	343	2401	16,807	117,649	823,543

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 Eq. 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 (Eq. 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 2: Solve collision.

```

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

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

```

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.

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 (Schulz et al. 2016) 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, Fig. 4 shows the

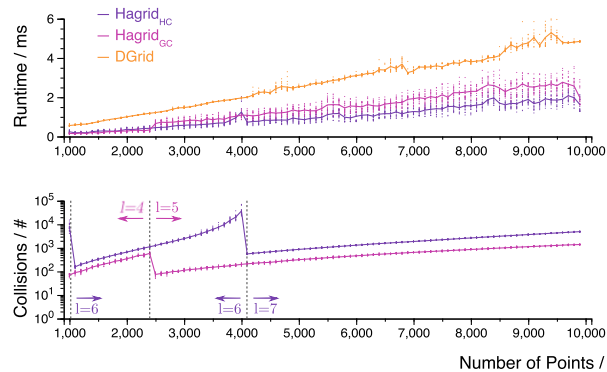


Fig. 4 Charts showing runtime evaluation of generated data (uniformly distributed points, with values from $[0, 1]$ for each dimension) with specific dataset sizes (x-axis) for Hagrid_{HC}, Hagrid_{GC}, and DGrid. We computed the grid layouts with the three techniques for dataset sizes from 1,000 to 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 $n = 2,400$)

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 $\mathcal{O}(n^2)$.

There is a relationship between our collision handling and linear probing in closed hashing (also called open addressing) (Cormen et al. 2009) that allows us to model an in-between case. If we assume that our data is distributed in a way that it leads to the same collision probabilities as in uniform hashing, then we arrive at an average number of probing steps $1/(1 - \alpha)$ for adding a point (or for unsuccessful search in hashing); here, α is the load factor, i.e., the percentage to which the table is filled by points. This means that the number of collision-handling steps depends on the load factor, but not on n . To fill the SFC, we keep on adding points subsequently, thus increasing the load factor from 0 to the maximum load factor α_{\max} at equidistant discretization steps.

In the limit case for asymptotic behavior, we then have infinitesimal steps for α , and $\int_0^{\alpha_{\max}} 1/(1 - \alpha) d\alpha = -\ln(1 - \alpha_{\max})$ for the average number of collision-handling steps. Therefore, with the assumption of uniform hashing and fixed maximum load factor, we have $c(n) \in \mathcal{O}(1)$ to handle one point and, thus, $\mathcal{O}(n \log n + n) = \mathcal{O}(n \log n)$ for the overall runtime, which is identical to the best case behavior. With increasing load factor, there is an increasing number of collisions, which impacts the runtime.

It should be noted that the input data at hand does not necessarily come with the properties of uniform hashing; therefore, the 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.

We introduce the quality metrics used, the setup of the evaluation, and the results and runtime analysis.

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. To this end, we have selected CMDS and DGRID as they specifically cater to scatterplot layout simplification. In previous evaluations against related methods (Eppstein et al. 2015; Fried

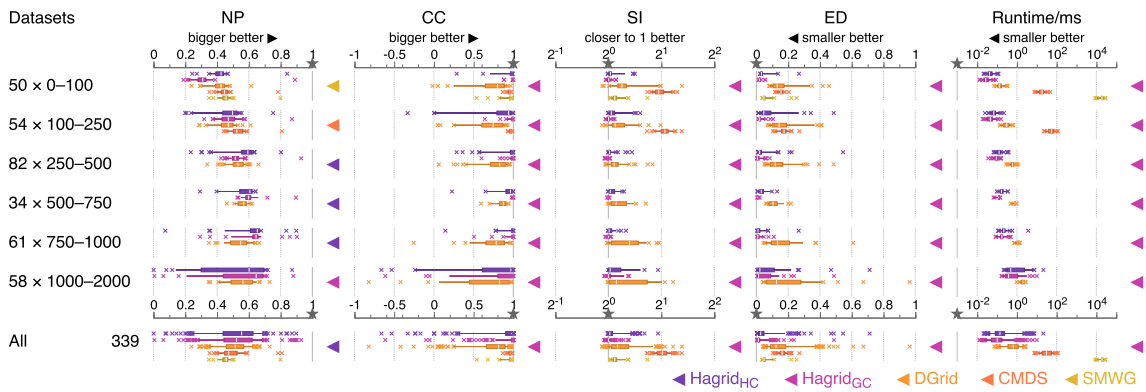


Fig. 5 Results of our evaluation of the 339 scatterplots, aggregated by dataset size, and with all data in the last row. The boxplots show the median, the 25% and 75% quantiles, while the ends show the 5- and the 95 percentile. The color of the triangles indicates the best method, measured by the median 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

et al. 2015; Strong and Gong 2014; Wood and Dykes 2008), 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 $\text{AUC}_{\log} \text{RNX}$ (Lee et al. 2015)), 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.

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 $[0, 1]$.

Neighborhood preservation (NP, or $\text{AUC}_{\log} \text{RNX}$ (Lee et al. 2015)) is a metric that enhances the metric k -Neighborhood Preservation (NP_k) (Paulovich and Minghim 2008) by aggregating the measurements for all neighborhood sizes k . The metric NP_k is used by nearly all methods mentioned in the related work section (Duarte et al. 2014; Gomez-Nieto et al. 2013a; Hilasaca and Paulovich 2019; Marcílio-Jr et al. 2019). It calculates the average percentage of the k -nearest neighbors of each box that are preserved in the final layout. It takes a value between 0 and 1.

We calculate NP_k as follows:

$$\text{NP}_k = \frac{1}{n \cdot k} \sum_{i=1}^n |V_i^{(k)}(X) \cap V_i^{(k)}(Y)|,$$

where X is the original set of points, Y is the gridified set of points. $V_i^{(k)}$ returns the set of the k -nearest neighbors of the i -th point of the respective set of datapoints, and n is the total number of points in the data. These values are scaled and aggregated as follows:

$$\begin{aligned} \widetilde{\text{NP}}_k &= \frac{(n-1)\text{NP}_k - k}{n-1-k}, \\ \text{NP} &= \frac{\left(\sum_{k=1}^{n-2} \frac{\widetilde{\text{NP}}_k}{k} \right)}{\left(\sum_{k=1}^{n-2} \frac{1}{k} \right)}. \end{aligned} \quad (2)$$

Values between 0 and 1 are possible, where 1 means perfect neighborhood preservation, and 0 means no neighborhood preservation.

Cross-correlation (CC) measures the distance correlation between pairwise distances in the original layout compared to ones in the new layout. The distance can be interpreted as a measure of dissimilarity between the points; ideally, dissimilar points in the original layout remain as such in the new one. Hilasaca and Paulovich (Hilasaca and Paulovich 2019) also use this measure in their evaluation of DGRID. The CC measure is defined as:

$$\text{CC} = \sum_{i=1}^n \sum_{j=1}^n \frac{(\delta(y_i, y_j) - \bar{\delta}_Y) \cdot (\delta(x_i, x_j) - \bar{\delta}_X)}{\sigma_X \cdot \sigma_Y}, \quad (3)$$

where x_i and x_j are points belonging to X (the original set of points), y_i and y_j are points belonging to Y (the gridified set of points), σ_X and σ_Y are the respective standard deviations, and $\bar{\delta}_X$ and $\bar{\delta}_Y$ are the respective mean distances between any pair of points.

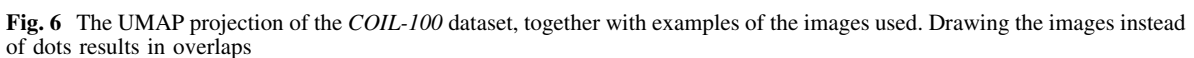
Euclidean distance (ED) is the average distance between the original points and their gridified counterparts:

where x_i and y_i correspond again to the original and the final position of our data points. We use Euclidean distance as a measure for the global structure of the visualization. The main objective of methods such as NMAP and DGRID is a space-filling visualization, rather than the preservation of global structure. Therefore, we expect them to perform poorly with respect to Euclidean distance. In our case, intuitively, in order for the global structure to change as little as possible, the points should be moved to a position as close as possible. The higher the average Euclidean distance is, the worse the global structure of the graph is preserved.

This measure is used for global structure preservation. Ideally, the resulting grid has a similar convex hull to the initial one. While the range for values of size increase is $]0, \infty[$, 1 is the optimal value. Size Increase has also been used in previous evaluations (Duarte et al. 2014; Gomez-Nieto et al. 2013b). Due to the space-filling aspect of DGRID, this metric will be expected to have worse results.

4.3 Evaluation data

For the evaluation, we gathered 60 real and synthetic datasets. 54 of these datasets stem from Aupetit and Sedlmair (Aupetit and Sedlmair 2016). 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 (Pearson 1901), robust PCA (Candès et al. 2011), t-SNE (van der Maaten and Hinton 2008), Isomap (Tenenbaum et al. 2000), LLE (Roweis and Saul 2000), UMAP (McInnes et al. 2018), Spectral Embedding (Belkin and Niyogi 2003), Gaussian Random Project (Bingham and Mannila 2001), and MDS (Kruskal 1964). For parameterizable DR methods, 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. For all tests, we have normalized the scatterplots so that the dimensions have values between 0 and 1.

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 (<http://www.gicentre.org/smwg/>) as we could not replicate the code based on publicly available information. For the evaluation we used the Python implementations,

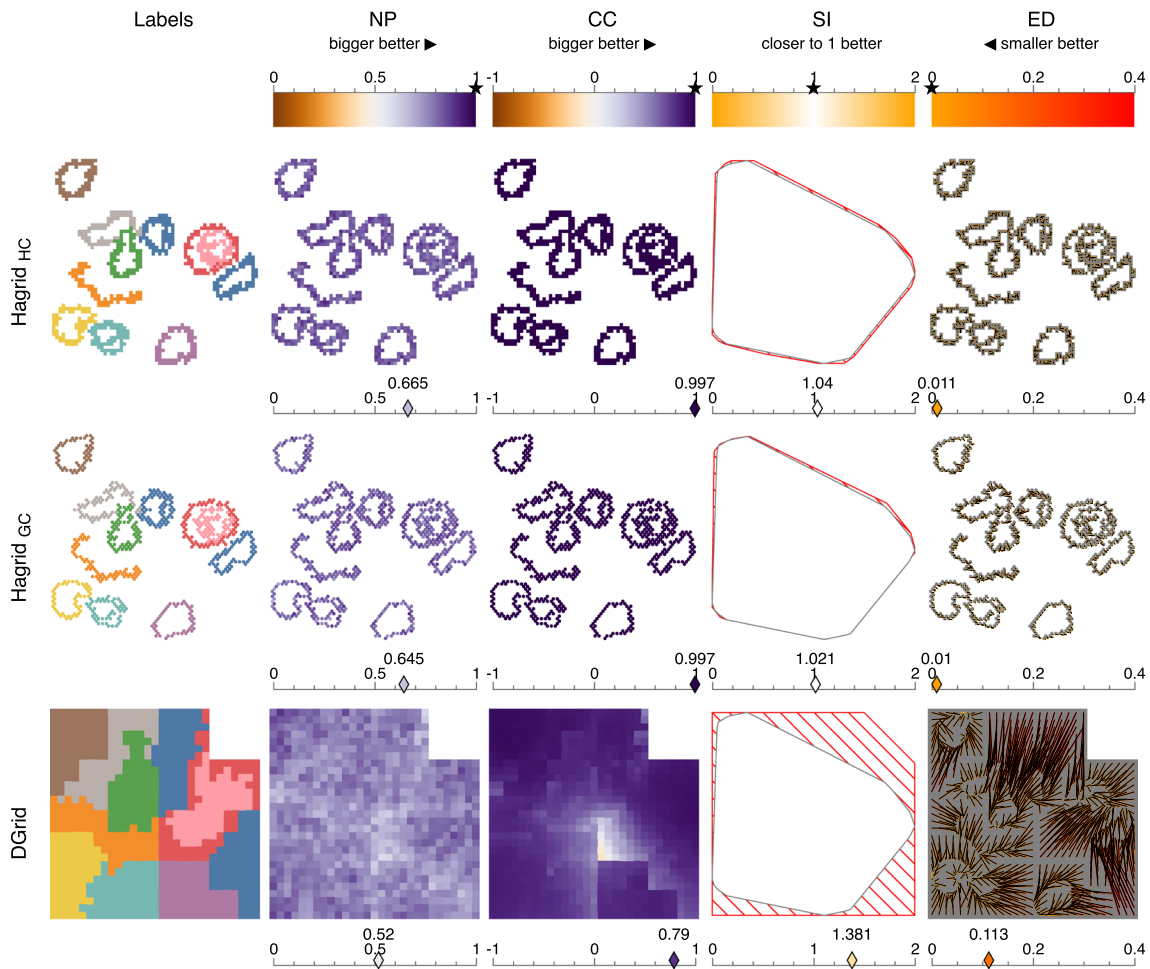


Fig. 7 Each row represents one of the evaluated techniques on the scatterplot shown in Fig. 6. The first column represents the resulting gridified scatterplot with dots color coded by labels. The next two columns plot each point in the grid layout color coded by the neighborhood preservation and cross-correlation metric. The last two columns show how the points moved from the original to the gridified layout in each case. The value underneath each visualization represents the overall metric calculated for the entire layout

because Python supports parallel processing for speeding up the computations of the evaluation metrics. 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 Fig. 5). 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.

For the HAGRID results, we used the level that guarantees at least 50% whitespace ($l_{50\%} = \lceil \log_m(n \cdot 1.5) \rceil$), to have space for maintaining global structures. If the vertices of the underlying SFC are completely occupied, the result would be space-filling as with DGRID. We used the recommended parameters for CMDS ($\alpha = 0.1$ and iter = 20 per round), for the size of one tile we used the size defined by Hagrid_{HC}, and set the boundaries to the extent of the scatterplot plus the size of one tile as margin. Furthermore, we fixed CMDS' number of maximum rounds to 50, so we could measure comparable runtime. DGRID allows one to set an aspect ratio, we computed square results as we normalized the scatterplot beforehand to a range of [0, 1] for all dimensions. For SMWG, we used the online tool, where we first loaded the dataset, then we defined the grid to have 0.5 whitespace and "minimum aspect ratio". For the optimization, we employed the simulated annealing option with default options but with weightings set to 1 for "VEC", "DIST", "TOPO", and "SHAPE", and the ones specifically important for maps ("DIR" and "DAT") to 0.

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

4.5 Results

Figure 5 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 Fig. 4. The supplemental material contains a more detailed version of Fig. 5, 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 Fig. 5, we use a \log_{10} -scale for runtime and a \log_2 -scale for the SI metric to improve readability.

To expand on the analysis provided by the boxplots in Fig. 5, we show in Fig. 7 the resulting grids for the COIL-100 dataset (Nene et al. 1996, only for HAGRID and DGRID because the runtime for CMDS was too

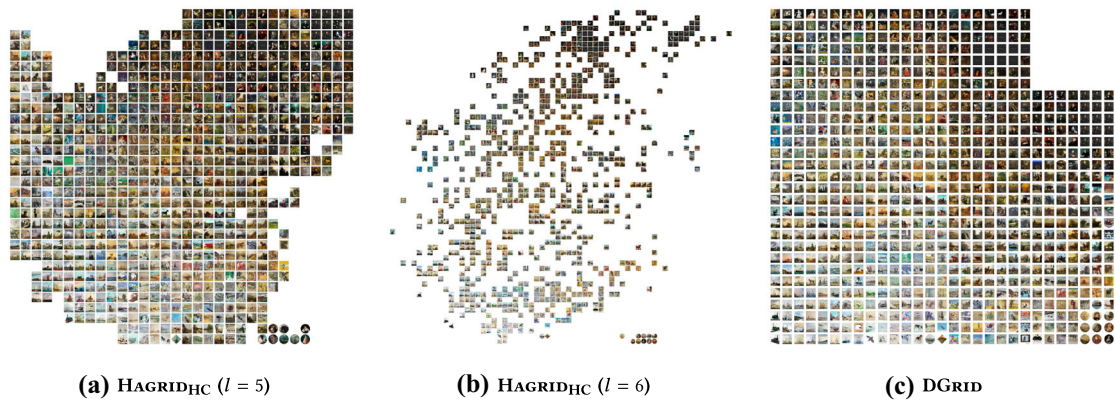


Fig. 8 Gridified versions of the t-SNE projection of the *Art UK Paintings* dataset (Crowley and Zisserman 2014) for different settings of the Hagrid_{HC} level parameters and the DGRID layout

high, and the online-tool for SMWG even crashed for this dataset. This data consists of 792 images of 10 random objects, for example a car, a rubber duck, and a toy photographed from many different angles (- see Fig. 6 left). We applied UMAP on this dataset, resulting in the scatterplot layout in Fig. 6. Our aim with selecting this dataset was both to highlight how different techniques transform such an intricate layout, and to provide an example of when preserving the global structure of a dataset might be beneficial.

In Fig. 7, points are colored according to labels or different quality metrics calculated for each of the individual points respectively. The space-filling approach DGRID preserves the local cluster structure. However, when there are more cells instantiated than points, DGRID places the empty cells on the borders of the grid rather than inside the clusters. This characteristic reduces the preservation of more global patterns.

From a qualitative angle, we see that DGRID is optimized for space-filling-ness, resulting in larger thumbnail image representations. If this is the sole objective, it is clearly a good choice. However, our goal was to additionally preserve spatial structures of the scatterplot. While DGRID also preserves the cluster contents, it loses information about the projection (e.g. potential manifolds), the particularity of the clusters, and the separation between them (Brehmer et al. 2014). The empty cells of DGRID are not optimally placed in this case.

5 Use cases

Our primary goal with HAGRID was to find a good balance between removing overlap and preserving scatterplot structure, while having a fast runtime. To illustrate the value of this combination, we now provide two use cases: adjusting the curve level to create different types of layouts, and realizing an interactive lens view.

5.1 Interactively adjusting level and point size

By altering the level parameter of our technique, HAGRID gives a natural handle to deal with the trade-off between space-filling-ness (cell size and space efficiency) and point position accuracy (global structure preservation). With a low level setting, HAGRID can achieve a more squared layout, similar to DGRID. An example is given in Fig. 8, where we can see the difference between a layout with the minimum $\text{Hagrid}_{\text{HC}}$ level (5), one level higher (6) that outputs the grid, and the DGRID layout for reference. In our situation, the white space is still scattered around the convex hull of the initial layout. Interactively transitioning between different levels allows to step-wise translate a scatterplot into a space-filling map with HAGRID.

5.2 Lens view

Due to its computational efficiency, our approach can also be used to support interactive lenses such as the one proposed in Glyphboard (Kammer et al. 2020). In Glyphboard, DR projections are plotted as normal

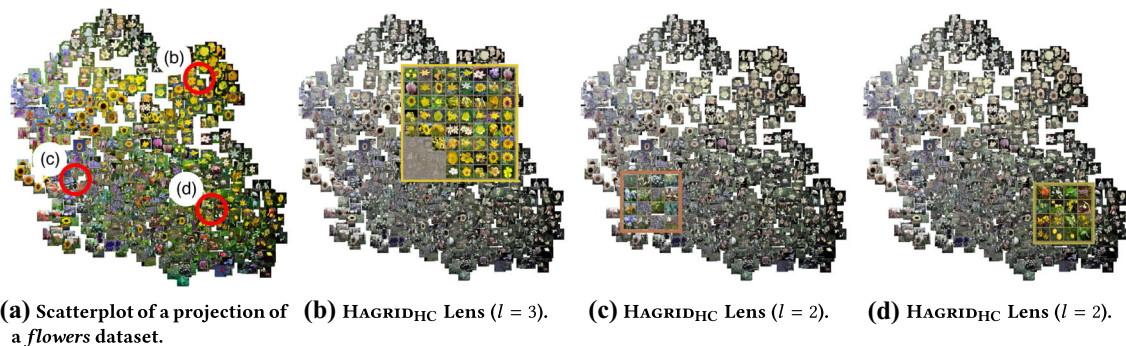


Fig. 9 Usage of HAGRID as lens. The circles indicate the mouse position for the screenshots (b), (c), and (d)

scatterplots and as the user zooms in, the dots are replaced with circular glyphs. The lens view in their implementation is using a force-directed layout algorithm to handle overlap, but that does not guarantee neighborhood preservation. Based on our comparison to CMD5, which closely resembles a force-directed layout algorithm, we believe our technique could fit this type of use case better. In Fig. 9, we show a possible way of using HAGRID as part of a lens view. Here, we used a t-SNE projection of photographs of flowers as the source dataset (Nilsback and Zisserman 2008). For the overall scatterplot, we maintain the original scatterplot, with overlaps. However, in the lens view we use a low-level HC, to dynamically align the photos. As our approach is computationally efficient, it lends itself toward supporting such interactive features.

6 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 Fig. 5). 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). A possible enhancement—reducing the sensitivity of HAGRID to extreme outliers—could be the use of adaptive SFC's (Zhou et al. 2020).

Like other comparative studies, our investigation is naturally limited to a certain selection of existing techniques. For future work, it might for instance also be interesting to compare to further other approaches. A recent update of DGRID (Hilasaca and Paulovich 2019) (version 4 on arXiv) adds dummy points to preserve also the global structure of a scatterplot. But similar to the extension of NMAP (Duarte et al. 2014) for producing a treemap with uniform rectangles, adding dummy points increases the runtime for gridifying a scatterplot. With some effort it is possible to disentangle the overlap removal method of the DUST & MAGNET (Vollmer and Döllner 2020) technique to use it as another overlap removal technique.

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 (Ghatak et al. 2013) 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.

Acknowledgements This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project-ID 251654672 - TRR 161.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

- Auber D, Huet C, Lambert A, Renoust B, Sallaberry A, Saulnier A (2013) GosperMap: using a gosper curve for laying out hierarchical data. *IEEE Trans Vis Comput Graph (TVCG)* 19(11):1820–1832. <https://doi.org/10.1109/TVCG.2013.91>
- Aupetit M, Sedlmair M (2016) SepMe: 2002 New visual separation measures. In: *IEEE Pacific Vis Symp (PacificVis)*. <https://doi.org/10.1109/PACIFICVIS.2016.7465244>
- Bader M (2012) Space-filling curves: an introduction with applications in scientific computing. Vol. 9. Springer Science & Business Media. <https://doi.org/10.1007/978-3-642-31046-1>
- Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput* 15(6):1373–1396. <https://doi.org/10.1162/089976603321780317>
- Bingham E, Mannila H (2001) Random projection in dimensionality reduction: applications to image and text data. In: *Proceedings of ACM international conference on knowledge discovery and data mining (SIGKDD)*. pp 245–250. <https://doi.org/10.1145/502512.502546>
- Borg I, Groenen P (2003) Modern multidimensional scaling: theory and applications. *J Educ Meas (JEM)* 4(3):277–280. <https://doi.org/10.1111/j.1745-3984.2003.tb01108.x>
- Brehmer M, Sedlmair M, Ingram S, Munzner T (2014) Visualizing dimensionally-reduced data: Interviews with analysts and a characterization of task sequences. In: *Proceedings of beyond time and errors: novel evaluation methods for visualization (BELIV)*. pp 1–8. <https://doi.org/10.1145/2669557.2669559>
- Buchmüller J, Jäckle D, Cakmak E, Brandes U, Keim DA (2018) MotionRugs: visualizing collective trends in space and time. *IEEE Trans Vis Comput Graph (TVCG)* 25(1):76–86. <https://doi.org/10.1109/TVCG.2018.2865049>
- Candès EJ, Li X, Ma Y, Wright J (2011) Robust principal component analysis? *J ACM* 58(3):37. <https://doi.org/10.1145/1970392.1970395>
- Cormen TH, Leiserson CE, Rivest RL (2009) *Introduction to algorithms*. MIT Press
- Crowley E, Zisserman A (2014) The state of the art: object retrieval in paintings using discriminative regions. In: *Proc. British Machine Vision Conf, BMVA Press*
- Cutura R, Morariu C, Cheng Z, Wang Y, Weiskopf D, Sedlmair M (2021) Hagrid: gridify scatterplots with hilbert and gosper curves. In: *International symposium on visual information communication and interaction*. pp 1–8. <https://doi.org/10.1145/3481549.3481569>
- Deng L (2012) The MNIST database of handwritten digit images for machine learning research. *IEEE Sign Process Mag* 29(6):141–142. <https://doi.org/10.1109/MSP.2012.2211477>
- Du Q, Faber V, Gunzburger M (1999) Centroidal voronoi tessellations: applications and algorithms. *SIAM Rev* 41(4):637–676. <https://doi.org/10.1137/S0036144599352836>
- Duarte FSLG, Sikansi F, Fatore FM, Fadel SG, Paulovich FV (2014) Nmap: a novel neighborhood preservation space-filling algorithm. *IEEE Trans Vis Comput Graph (TVCG)* 20(12):2063–2071. <https://doi.org/10.1109/TVCG.2014.2346276>
- Dwyer T, Marriott K, Stuckey PJ (2006) Fast node overlap removal-correction. In: *International symposium on graph drawing*. Springer, pp 446–447. https://doi.org/10.1007/978-3-540-70904-6_44
- Eppstein D, van Kreveld M, Speckmann B, Staals F (2015) Improved grid map layout by point set matching. *Int J Comp Geom Appl* 25(2):101–122. <https://doi.org/10.1142/S0218195915500077>
- Fried O, DiVerdi S, Halber M, Sizikova E, Finkelstein A (2015) IsoMatch: creating informative grid layouts. In: *Computer graphics forum*, Vol. 34. Wiley Online Library, pp 155–166. <https://doi.org/10.1111/cgf.12549>
- Gansner ER, Hu Y (2008) Efficient node overlap removal using a proximity stress model. In: *International Symposium on Graph Drawing*. Springer, pp 206–217. https://doi.org/10.1007/978-3-642-00219-9_20
- Gardner M (1976) Mathematical games-in which “monster” curves force redefinition of the word “curve”. *Sci Am* 235(6):124–133
- Ghatak R, Pal M, Goswami C, Poddar DR (2013) Moore curve fractal-shaped miniaturized complementary spiral resonator. *Microw Opt Technol Lett* 55(8):1950–1954. <https://doi.org/10.1002/mop.27682>
- Gomez-Nieto E, Casaca W, Nonato LG, Taubin G (2013) Mixed integer optimization for layout arrangement. In: *Symposium on graphics, patterns and images (SIBGRAPI)*. IEEE, pp 115–122. <https://doi.org/10.1109/SIBGRAPI.2013.25>
- Gomez-Nieto E, Roman FS, Pagliosa P, Casaca W, Helou ES, de Oliveira MCF, Nonato LG (2013) Similarity preserving snippet-based visualization of web search results. *IEEE Trans Vis Comput Graph (TVCG)* 20(3):457–470. <https://doi.org/10.1109/TVCG.2013.242>
- Hilasaca G, Paulovich FV (2019) Distance preserving grid layouts. *arXiv preprint arXiv:1903.06262*
- Hilbert D (1935) Über die stetige Abbildung einer Linie auf ein Flächenstück. In: *Dritter band: analysis Grundlagen der mathematik physik verschiedenes*. Springer, pp 1–2. https://doi.org/10.1007/978-3-662-38452-7_1
- Jeżowicz T, Gajdoš P, Ochodková E, Snášel V (2014) A New iterative approach for finding nearest neighbors using space-filling curves for fast graphs visualization. In: *International Joint Conference SOCO’14-CISIS’14-ICEUTE’14*. Springer, pp 11–20. https://doi.org/10.1007/978-3-319-07995-0_2
- Kammer D, Keck M, Gründer T, Maasch A, Thom T, Kleinstenuber M, Groh R (2020) Glyphboard: visual exploration of high-dimensional data combining glyphs with dimensionality reduction. *IEEE Trans Vis Comput Graph (TVCG)* 26(4):1661–1671. <https://doi.org/10.1109/TVCG.2020.2969060>
- Keim DA, Herrmann A (1998) The Gridfit algorithm: an efficient and effective approach to visualizing large amounts of spatial data. In: *Proc. Visualization*. pp 181–188. <https://doi.org/10.1109/VISUAL.1998.745301>
- Kruskal JB (1964) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29(1):1–27. <https://doi.org/10.1007/BF02289565>
- Lee JA, Peluffo-Ordóñez DH, Verleysen Michel (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>

- Liu X, Hu Y, North S, Shen H-W (2018) CorrelatedMultiples: spatially coherent small multiples with constrained multi-dimensional scaling. In: Computer graphics forum, Vol. 37. Wiley Online Library, pp 7–18. <https://doi.org/10.1111/cgf.12526>
- Marcelio-Jr WE, Eler DM, Garcia RE, Pola IRV (2019) Evaluation of approaches proposed to avoid overlap of markers in visualizations based on multidimensional projection techniques. *Inf Vis* 18(4):426–438. <https://doi.org/10.1177/1473871619845093>
- McInnes L, Healy J, Melville J (2018) UMAP: uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*
- McNeill G, Hale SA (2017) Generating tile maps. In: Computer graphics forum, Vol. 36. Wiley Online Library, pp 435–445. <https://doi.org/10.1111/cgf.13200>
- Meulemans W, Dykes J, Slingsby A, Turkay C, Wood J. Small multiples with gaps. <http://www.gicentre.org/smwg/>. Accessed 1 Dec 2020
- Meulemans W, Dykes J, Slingsby A, Turkay C, Wood J (2016) Small multiples with gaps. *IEEE Trans Vis Comput Graph (TVCG)* 23(1):381–390. <https://doi.org/10.1109/TVCG.2016.2598542>
- Meulemans W, Sondag M, Speckmann B (2020) A simple pipeline for coherent grid maps. *IEEE Trans Vis Comput Graph (TVCG)*. <https://doi.org/10.1109/TVCG.2020.3028953>
- Muelder C, Ma K-L (2008) Rapid graph layout using space filling curves. *IEEE Trans Vis Comput Graph (TVCG)* 14(6):1301–1308. <https://doi.org/10.1109/TVCG.2008.158>
- Nachmanson L, Nocaj A, Bereg S, Zhang L, Holroyd A (2016) Node overlap removal by growing a tree. In: International symposium on graph drawing and network visualization, Springer, pp 33–43. https://doi.org/10.1007/978-3-319-50106-2_3
- Nene SA, Nayar SK, Murase H et al. (1996) Columbia object image library (coil-20)
- Nilsback M-E, Zisserman A (2008) Automated flower classification over a large number of classes. In: *Conference on Computer Vision, Graphics and Image Processing*. IEEE, pp 722–729. <https://doi.org/10.1109/ICVGIP.2008.47>
- Paulovich FV, Minghim R (2008) HiPP: a novel hierarchical point placement strategy and its application to the exploration of document collections. *IEEE Trans Vis Comput Graph (TVCG)* 14(6):1229–1236. <https://doi.org/10.1109/TVCG.2008.138>
- Pearson K (1901) LIII. On lines and planes of closest fit to systems of points in space. *London Edinb Dublin Philos Mag J Sci* 2(11):559–572. <https://doi.org/10.1080/14786440109462720>
- Pinho R, de Oliveira MCF, de Lopes AA (2009) Incremental board: a grid-based space for visualizing dynamic data sets. In: *ACM Symposium on Applied Computing (SAC)*. pp 1757–1764. <https://doi.org/10.1145/1529282.1529679>
- Quadrianto N, Smola AJ, Song L, Tuytelaars T (2010) Kernelized sorting. *IEEE Trans Patt Anal Mach Intell* 32(10):1809–1821. <https://doi.org/10.1109/TPAMI.2009.184>
- Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500):2323–2326. <https://doi.org/10.1126/science.290.5500.2323>
- Schulz C, Nocaj A, El-Assady M, Frey S, Hlawatsch M, Hund M, Karch G, Netzel R, Schätzle C, Butt M, Keim DA, Ertl T, Brandes U, Weiskopf D (2016) Generative data models for validation and evaluation of visualization techniques. In: *Proceedings of beyond time and errors: novel evaluation methods for visualization (BELIV)*. pp 112–124. <https://doi.org/10.1145/2993901.2993907>
- Sedlmair M, Tatu A, Munzner T, Tory M (2012) A taxonomy of visual cluster separation factors. In: *Computer graphics forum*, Vol. 31. Wiley Online Library, pp 1335–1344. <https://doi.org/10.1111/j.1467-8659.2012.03125.x>
- Strobelt H, Spicker M, Stoffel A, Keim D, Deussen O (2012) Rolled: out Wordles—a heuristic method for overlap removal of 2D data representatives. In: *Computer graphics forum*, Vol. 31. Wiley Online Library, pp 1135–1144. <https://doi.org/10.1111/j.1467-8659.2012.03106.x>
- Strong G, Gong M (2014) Self-sorting map: an efficient algorithm for presenting multimedia data in structured layouts. *IEEE Trans Multimed* 16(4):1045–1058. <https://doi.org/10.1109/TMM.2014.2306183>
- Tenenbaum JB, De Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500):2319–2323. <https://doi.org/10.1126/science.290.5500.2319>
- Vojtěch U, Petr G, Václav S, Yu-Chi L, Michal Radecký (2019) Hierarchical hexagonal clustering and indexing. *Symmetry* 11(6):731. <https://doi.org/10.3390/sym11060731>
- van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. 9, pp 2579–2605. <http://jmlr.org/papers/v9/vandermaaten08a.html>
- Vollmer JO, Döllner J (2020) 2.5D Dust and magnet visualization for large multivariate data. In: *International symposium on visual information communication and interaction*. pp 21–1. <https://doi.org/10.1145/3430036.3430045>
- Ward MO (2002) A taxonomy of glyph placement strategies for multidimensional data visualization. *Inf Vis* 1(3–4):194–210. <https://doi.org/10.1057/PALGRAVE.IVS.9500025>
- M Wattenberg (2005) A note on space-filling visualizations and space-filling curves. In: *Proceedings of the IEEE information visualization symposium*. IEEE, pp 181–186. <https://doi.org/10.1109/INFVIS.2005.1532145>
- Wilkinson L, Anand A, Grossman R (2005) Graph-theoretic scagnostics. In: *Proceedings of the IEEE information visualization symposium*. IEEE, pp 157–164. <https://doi.org/10.1109/INFVIS.2005.1532142>
- Wood J, Dykes J (2008) Spatially ordered treemaps. *IEEE Trans Vis Comput Graph (TVCG)* 14(6):1348–1355. <https://doi.org/10.1109/TVCG.2008.165>
- Zhou L, Johnson CR, Weiskopf D (2020) Data-driven space-filling curves. *IEEE Trans Vis Comput Graph (TVCG)*. <https://doi.org/10.1109/TVCG.2020.3030473>