# Algorithms for coherent rectangular visualizations

*Citation for published version (APA):*
Sondag, M. F. M. (2020). *Algorithms for coherent rectangular visualizations*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven.

*Document status and date:*
Published: 30/11/2020

*Document Version:*
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

*Please check the document version of this publication:*

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

*General rights*
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

*Take down policy*
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Nov. 2023

# Algorithms for Coherent Rectangular Visualizations

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen
op maandag 30 november 2020 om 13:30 uur

door

Max Franciscus Maria Sondag

geboren te Uden

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter: prof.dr. M.T. de Berg

promotor: prof.dr. B. Speckmann

copromotor: dr. W. Meulemans

leden: dr. K.A. Buchin
dr. K.A.B. Verbeek
prof.dr. A. Schulz (Fernuniverstität Hagen)
prof.dr. J.D. Wood (City University of London)

*Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.*

The work in this thesis is supported by the Netherlands' Organization for Scientific Research (NWO) under project no. 639.023.208.

The work in the thesis has been carried out under the auspices of the research school IPA (Institute for Programming Research and Algorithmics).

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5139-2.

Printed by Ipskamp Printing, Enschede.

© 2020 by Max Franciscus Maria Sondag. All rights reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

# Acknowledgments

I had not truly considered to do a PhD before Bettina asked me whether I would be interested four years ago. After four years of doing a PhD however, I am happy that I took the opportunity when it presented itself. It has been a wonderful journey through the academic world, and if I could go back in time I would do it all over again. I would like to take this opportunity to thank everyone that supported me.

First of all, I want to thank my supervisors Bettina and Wouter. As my longest lasting supervisor, Bettina has been supervising me starting from the masters project up to the very end of this PhD. I enjoyed our weekly in-person meetings with all the academic and collateral knowledge that I obtained from them. Although meetings sometimes got shifted by the need of the ever-overflowing agenda, Bettina always managed to ensure that somewhere there a spot was available for me. Also, although not always going as planned, I greatly enjoyed working with the interesting collaborators she introduced me to who worked in completely different domains. While Wouter only joined later as my supervisor, he has not been any less helpful to me. He always kept the door open for discussion, and I have fond memories of chalking many whiteboards with possible designs and ideas. Both of them have been immensely helpful to me and I will not forget about them anytime soon.

Writing this thesis in quarantine was not an easy task. It made me realize how much I missed the contact with all the people around me. I would like to thank, and possibly apologize to, Wiebe for keeping up with my thesis-induced insanity. Having someone physically there to vent to is a boon not to be underestimated. I would also like to thanks my parents for the wonderful idea of taking a chair to the park and writing my thesis there, instead of being stuck between the 4 walls of my room with a glowing computer screen. My parents have not only helped and supported me during COVID-19, but during my entire life, pushing me where I needed and wanted to go and supporting me whenever it was needed, and I couldn't be more thankful for that. It was not always easy, but I ended up in a better position due to it. I would also like to thank my brother, Thijs, for broadening my a view of what is possible. I do not think I would consider half of what I am doing if I had not seen all your ideas. Also many thanks to my sister, Marit, for letting me keep a view on what is important in life.

Throughout my PhD I enjoyed the company of many different friends, and I would like to take this opportunity to thank them for keeping my mind occupied with things other than work. To Ruud, Rik, Jasper, Teun and Hein whom I have known since the start of my studies at this university, thank you for the many fun evening and trips. We created many memories that I will never forget. I would also like to thank Dirk, Luuk and Rogier. We have known each other for a long long time already, and yet I still enjoy the moments that we get together again. Many thanks to Arthur, Aleks, Adam, Marcel, and Quirijn as well for dragging me into the world of DnD, and to Jennifer, Jay, Tom, Simon, Laura, Annelies, Roderik, Kevin, and Merlijne for keeping me there and having many an enjoyable night. I would also like to thank Arthur V, Dion, Jan-Willem, Niels, Rob, Patrick, and many others with whom I had the pleasure to spend the night bending our heads over a board game or having a drink with in the last 4 years.

I would also like to thank the members of my first office. Ali, Max, Thom, Dániel, and Huib (and Ferry), I've had a great time having you as my office mates. Not only were you helpful as sparring partners for research, but I also had a lot of fun in the office. We talked, made whiteboard art, discussed and ranted more than I should admit here, but I enjoyed all of it. Although I had an official office, I was also often present in my unofficial secondary office which became my home after one meeting and chat too much with its occupants. Thanks to Quirijn, Tim, Willem, Jules, Bram, Pantea, and very briefly Nathan and Tom as well for making my life so much more enjoyable, and apologies for stealing all your whiteboards. I would like to thank Jules in particular for introducing me to bouldering, which gave me a physical instead of a mental wall to hit my head against.

I had many fun trips and workshops with my colleagues from the AGA research group. While we ended up wet from either the rain or the river most of the time, I enjoyed the cycling, hiking, dining, and other activities and it was always a welcome change of pace. Thanks to Bettina, Kevin V, Wouter, Irina, Arthur, Marcel, Ignaz, Quirijn, Tim, Thom, Willem, Irene, Jules, Bram, Pantea, Nathan, Tom, Riet, and Agnes for forming a great and enjoyable group. Also thanks to the members of the ALGO group Aleks M., Aleks P., Ali, Astrid, Bart, Dániel, Henk, Herman, Huib, Jari, Kevin B., Leyla, Mark, Martijn, Max K., Mehran, Michał, Morteza, Sándor, Shivesh, and Sudeshna who often joined us for lunch and other activities, and with whom I could often had enjoyable discussions about absolutely nothing important.

I've had the fortune to work with many different people during my time as a PhD-student and I would like to thank my co-authors Wouter, Bettina, Kevin V, Irina, Maarten L, Willem, Jules, Arthur, Marc, Daniel, Christoph, Maarten N, Soeren, Markus, Jaakko, Martijn, Emily, Ignaz, Mark, Scott, Alexandru, Eduardo, João, for their collaboration on various projects. Finally, I would like to thank Andre Schulz, Kevin Buchin, Kevin Verbeek, and Jo Wood for reviewing my thesis.

# Contents

**1 Introduction** **1**
- 1.1 Coherence in data . . . 3
- 1.2 Rectangular visualizations . . . 6

**2 Time-varying Treemaps** **11**
- 2.1 Rectangular treemaps . . . 14
- 2.2 Layouts . . . 17
- 2.3 Local moves algorithm . . . 19
  - 2.3.1 Transforming rectangular layouts using local moves . . . 20
  - 2.3.2 Algorithm . . . 21
  - 2.3.3 Improving performance . . . 24
- 2.4 Metrics . . . 25
  - 2.4.1 Visual quality . . . 26
  - 2.4.2 Stability . . . 26
- 2.5 Data . . . 30
  - 2.5.1 Data features . . . 30
  - 2.5.2 Data classes . . . 32
  - 2.5.3 Datasets . . . 33
- 2.6 Experimental results . . . 35
  - 2.6.1 Data classification analysis . . . 36
  - 2.6.2 Performance analysis across features . . . 38
  - 2.6.3 Comparison of data classes . . . 43
- 2.7 Discussion and conclusion . . . 45

**3 Uncertainty Treemaps** **49**
- 3.1 Related work . . . 51
- 3.2 Hierarchical uncertainty masks . . . 54
  - 3.2.1 Mask design . . . 54
  - 3.2.2 Reading an uncertainty treemap . . . 55
  - 3.2.3 Expert review . . . 56
  - 3.2.4 Mask quality . . . 60
- 3.3 Algorithms for uncertainty treemaps . . . 61

3.3.1 Mask-friendly algorithms . . . . . . . . . . . . . . . . . . 63
3.3.2 Mask-aware algorithms . . . . . . . . . . . . . . . . . . . 64
3.4 Experimental results . . . . . . . . . . . . . . . . . . . . . . . . 66
3.4.1 Relating mask-quality metrics . . . . . . . . . . . . . . . . 67
3.4.2 Effect of mask-friendly algorithms . . . . . . . . . . . . . 68
3.4.3 Effect of mask-aware algorithms . . . . . . . . . . . . . . 70
3.5 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
3.5.1 Mask shape and placement . . . . . . . . . . . . . . . . . 73
3.5.2 Mask rendering . . . . . . . . . . . . . . . . . . . . . . . 74
3.5.3 Uncertainty and hierarchy . . . . . . . . . . . . . . . . . . 76
3.6 Conclusion and future work . . . . . . . . . . . . . . . . . . . . 78

**4 A Simple Pipeline for Grid Maps** **79**
4.1 Problem exploration . . . . . . . . . . . . . . . . . . . . . . . . 80
4.1.1 Facets of coherence . . . . . . . . . . . . . . . . . . . . . 81
4.1.2 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . 83
4.2 A 3-step pipeline for coherent grid maps . . . . . . . . . . . . . 88
4.3 Decompose shape based on salient features . . . . . . . . . . . . 89
4.4 Arrange tiles using mosaic cartograms . . . . . . . . . . . . . . 92
4.5 Assign elements to tiles via point-set matching . . . . . . . . . . 94
4.6 Results and discussion . . . . . . . . . . . . . . . . . . . . . . . 95
4.7 Conclusions and future work . . . . . . . . . . . . . . . . . . . . 99

**5 Spatial set visualization** **103**
5.1 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104
5.2 Characterizing colored grids with a painting . . . . . . . . . . . 105
5.2.1 Simple purple regions . . . . . . . . . . . . . . . . . . . . 106
5.2.2 Spiderweb gadgets . . . . . . . . . . . . . . . . . . . . . . 108
5.2.3 Purple regions with holes . . . . . . . . . . . . . . . . . . 111
5.3 Optimizing panels . . . . . . . . . . . . . . . . . . . . . . . . . . 114
5.3.1 Ensuring a 5-painting . . . . . . . . . . . . . . . . . . . . 115
5.3.2 Ensuring a 2-painting . . . . . . . . . . . . . . . . . . . . 117
5.4 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121

**6 Conclusion** **123**
6.1 Main results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
6.2 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 124

**Bibliography** **127**

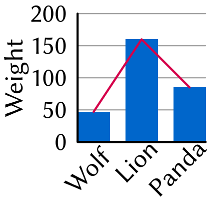
**Summary** **141**

**Curriculum vitae** **143**

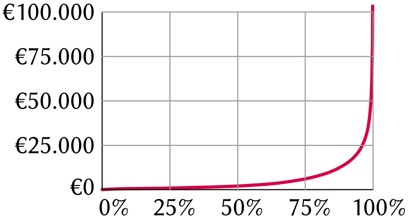
# Chapter 1

# Introduction

The amount of data that chronicles our life, work and world has rapidly increased in the past decades. To gain insight into these large quantities of data, we often turn to visualization techniques to use our powerful perceptual system for (parts of) the analysis which allows us to be more effective [90]. When designing visualizations we have to take both the type of data and the task to be performed into account to ensure that the visualization is effective. For example, a line chart is an effective tool to visualize continuous time-varying data such as the weight of a person over time. However, it is not suitable for categorical data such as the weight of different animal species, as the interpolated values do not have meaning. In contrast, a bar chart would better support the comparison between the weights of different species, as it does not suggest unnecessary interpolation between animal species. It would however be a less effective tool to visualize the weight of a person where this interpolation matches the continuous nature of the change in weight.



The efficacy of a visualization similarly depends on the task. Imagine that we have data that shows the average income for each country in the world. If the viewer is interested only in knowing the distribution of average incomes and is not interested in the spatial dimension of the data, a distribution chart is an effective to tool visualize it. However, if the viewer wants to investigate the spatial dimension of the data the visualization is no longer effective at leveraging the perceptual system, as this dimension is not visible in a distribution chart.



For a visualization to be effective, it has to visually represent the properties of the data that the viewer requires for the task. A visualization represents these properties by mapping them to visual variables such as length, position, color, and so on. Care should be taken to ensure that this mapping results in a visualization that is faithful to the underlying data such that the viewer sees a factual representation of the data. There are multiple definitions to define if a visualization is faithful and they can be roughly divided into two categories.

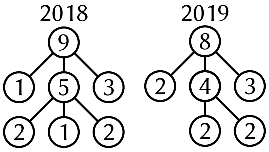
The first category defines faithfulness from the perspective of the data. Sugibuchi, Spyratos and Siminenko [129] argue that the mapping from the data to the visual variables should be an injective function. In addition, they require that relations between the properties of the data should be reflected in the visualization. Nguyen and Eades [93] propose three different kinds of faithfulness, two of which lie in this category: information faithfulness and change faithfulness. In the former, all properties in the data must be visually represented in the visualization. In the latter, change in the data should be consistent with change in the visual representation. Change faithfulness is also known as the principle of unambiguous data depiction [68]. Note that this category of faithfulness definitions does not take into account how the visualization is perceived, but focuses on the mapping from the data to the visualization.

The second category of faithfulness definitions defines faithfulness from the perspective of the visualization. This category contains the third definition of faithfulness that Nguyen and Eades propose is found: task faithfulness. A visualization is task faithful if it is accurate enough to correctly perform tasks. Thus, not only must the information be present, it needs to be accurately perceivable by the viewer. Tufte [140] argues that in a faithful visualization the size of the visual representation should match the true proportions of numerical data. That is, a larger visual representation should represent a higher numerical value. Mittelstädt et al. [89] propose a slightly different definition arguing that in a faithful visualization what the viewer sees should perceptually match to what is in the data. The visualization should thus compensate for physiological biases, possibly distorting the visual representation. Finally, Kim et al. [67] argue that visually related items should be related in the data.

In this thesis we focus on the *coherence* between the relations in the data and the visually perceived relations which relates to both categories of faithfulness. In a coherent visualization, relations between data items are visually represented, and visible relations are present in the data. A coherent visualization thus prevents false patterns from emerging in the visualization and preserves patterns that exist in the data. We thus consider both the mapping from the data to the visualization as well as the correspondence from the visualization to the data.

## 1.1 Coherence in data

Coherence can be expressed and measured in different ways depending on the data type. In this thesis we focus on coherent visualizations for four different data types: *(1)* time-varying hierarchical data, *(2)* uncertain hierarchical data, *(3)* geospatial data, and *(4)* spatial set data.



**Time-varying hierarchical data** impose an explicit hierarchy on the data items. This hierarchy can be represented by a tree where each data item represents a node in the tree. Hierarchical datasets are quite common, as data items are often clustered and subdivided when the number of items is large. The temporal dimension of hierarchical data naturally arises when collecting data from the same source multiple times. Both the value of the data items as well as the hierarchy can change over time. An example of a time-varying hierarchical dataset is the expenditure budget of a country. The expenses naturally form a hierarchy based on the different categories of expenses such as health care, road maintenance, etc., and the amount spent on each of these categories changes every year.

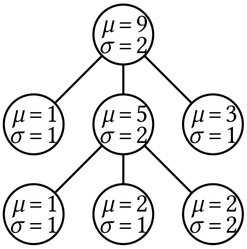
One common technique to visualize time-varying data is to visually represent each time step using the same visualization technique separately. The changes in the data are then visually represented by the change in the visualization. While numerous techniques for showing data of a single time step exist, they are not all equally suitable for time-varying data. Such techniques may result in a large change in the visualization for only a small change in the data. This may result in the viewer incorrectly concluding that large changes occur in the data over time.

To ensure coherence in the temporal dimension, the visualization needs to be coherent over time or *stable*[1]. That is, small changes in the data should result in small changes in the visualization. Such stable behavior ensures that the only changes the viewer sees are due to the data and thus the viewer can relate the visual change to the magnitude of change in the data. An additional advantage of having a *stable* visualization is that the mental map of the viewer between time steps is maintained. When items are visualized in a stable manner it becomes easier for the viewer to find and keep track of the data items [4].

---

[1] Kindlmann and Shedeidegger [68] proposed a general design rule using a similar concept.

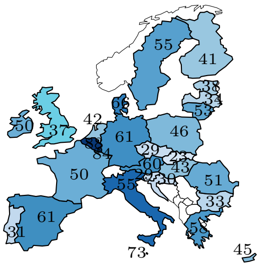
**Uncertain hierarchical data** associate uncertainty with each numerical data value. The data values are then estimations and not exact values. For example, any inference about a value in the future is necessarily imprecise, and similarly any physical measurement has only bounded accuracy. In uncertain hierarchical data, each interior node has uncertainty arising due to upwards propagation of uncertainty or due to inherent uncertainty of the value of the node. In order to faithfully represent such data, the uncertainty should be visually represented for each data item in the hierarchy.



For low-complexity data, a common way to visualize the uncertainty is using error bars, for example in line or bar charts. These error bars are closely linked to the numerical value in the visualization and give the viewer an immediate understanding of the uncertainty associated with the value. For more complex data, error bars cannot always be applied and alternatives are required.

To ensure a coherent visualization of the relation between data and uncertainty, the uncertainty and the data should be tightly linked in the visualization. Thus, the representation of the uncertainty of a data item should be graphically close to the representation of the numerical value of that data item. Moreover, the uncertainty should be visually quantifiable in such a way that it can be directly related to the data value. Such coherence between data value and uncertainty enables viewers to be aware of and estimate the influence of the uncertainty.

**Geospatial data** are a common data type where a spatial dimension, often location, is associated with each data item. The voter turnout per country in Europe for the European Parliament is an example of one such dataset. Here, each data item contains not only a value reflecting the percentage of eligible voters that voted, but additionally has a country region attached to it. To faithfully visualize this data the visualization should reflect the additional spatial dimension.

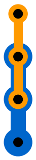


Most commonly, the spatial dimension is represented by showing data on the map itself, traditionally the realm of thematic cartography. But as data complexity rises, maps that place information at an exact spatial location become unreadable, as visual elements necessarily become cluttered. Hence, some form of schematization, that is, controlled and deliberate distortion of the spatial dimension, is generally necessary to support visualizations of complex data.

Viewers are often familiar with the geographic area visualized and thus have an intuitive idea where elements are supposed to be. In a map of Europe, Spain should for example be in the south-west, and the viewer expects France to be north-east of Spain. An effective visualization thus has to be coherent with the underlying geographical space. There are many different facets to this spatial coherence such as preserving adjacencies, distances or directions between the elements. Depending on the task, different facets are relevant to the viewer. For an effective visualization, the schematization thus needs to ensure that the relevant facets of spatial coherence for the task are not distorted.

**Spatial set data** define a number of set relations on data with a spatial dimension. An example of such a dataset are the characteristics of European cities. Each city has a spatial location and a number of characteristics such as whether it has an airport, whether it is a capital, whether it is industrial, and so on. To ensure that patterns in the data can be uncovered, the visualization needs to be coherent with both the spatial dimension as well as the relations between the sets themselves.

Visualizing spatial set data is equivalent to visualizing hypergraphs with fixed node positions. A hypergraph generalizes a graph by allowing an edge to connect to any number of nodes; such a hyperedge readily define a set. A common way to visualize spatial hypergraphs is to represent each node as a point on a map and show a hyperedge as a polygon that overlaps the associated points. We can identify the hyperedges that a node belongs to by considering which polygons it is contained in. Current techniques usually allow the polygons to overlap at the nodes, but also at other places. Depending on the visual representation this may result in the sets being locally nested. However, nesting may give a strong visual cue of containment implying a relation between the sets that might not be present. Nesting can thus result in incoherence between the relations implied in the visualization and the relations between the sets in the data.



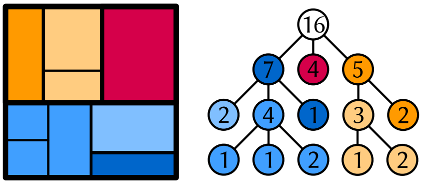
To ensure that the relations between sets are visualized in a coherent manner we should not imply containment and other relations between the sets when they are not present. Additionally, due to the spatial dimension of the data the visualization needs to ensure spatial coherence such that spatial patterns are reflected faithfully as well.

## 1.2 Rectangular visualizations

Many visualizations use rectangles in one form or another. Bar charts use rectangles to show the value of a data item, calendars use a rectangle to show a day, and text and labels are usually drawn in rectangular boxes for ease of reading. Rectangles have visually simple shapes which result in low visual complexity in the visualization. In addition, they can tile the plane making it easier for algorithms to use all the available screen space and thus avoid unnecessary whitespace.

Our focus in this thesis lies on rectangular visualizations, that is, visualizations that use rectangles as their fundamental building blocks. We describe several new algorithms that transform the data types described above into coherent rectangular visualizations. We visualize *(1)* time-varying hierarchical data and *(2)* uncertain hierarchical data with treemaps. We visualize *(3)* geospatial data with grid maps, and finally, we visualize *(4)* spatial set data using disjoint colored polygons.

### Treemaps



A well-established method for visualizing large hierarchical datasets are treemaps. Given a hierarchical dataset represented by a tree, treemaps recursively partition a 2D spatial region into cells whose visual attributes (area, color, shading, or annotation) encode the tree's data attributes. Compared to other methods such as node-link techniques, treemaps use all available screen pixels to show data, and thus can display trees of tens of thousands of nodes on a single screen. In this thesis we focus on rectangular treemaps, that partition a rectangular region into rectangles.

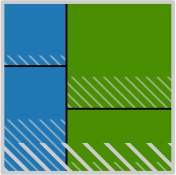
An effective treemap has high *visual quality*. That is, it is easy to determine the corresponding data value for each rectangle by estimating its area. The most common way to determine the visual quality is via the aspect ratio of the rectangles in the treemap. Kong *et al.* [70] show that large aspect ratios make it more difficult to estimate the area of a rectangle and to compare nodes with each other. Thus for a treemap to have a high visual quality we need to ensure that the rectangles have low aspect ratio. While optimizing the visual quality is NP-hard [10], there are a variety of treemapping algorithms that result in treemaps with high visual quality in practice. However, when the data changes over time these treemapping algorithms are not always coherent with the temporal dimension: small changes in the data can result in large changes in the treemap.

**Contribution** In Chapter 2 we study the problem of maintaining coherence for time-varying hierarchical data in treemaps. We propose a new state-aware algorithm that changes the layout of an existing treemap using only local moves. Local moves are small, local modifications to the treemap. By applying only local moves to the layout we can maintain the temporal coherence as the data changes, while having enough flexibility to obtain high visual quality. In contrast to existing treemapping algorithms which can reach only sliceable layouts, the full range of rectangular treemap layouts can be explored using local moves.

To better quantify temporal coherence we propose a new method to measure the stability of time-varying treemaps which explicitly considers the change in the input data. To this end, we introduce baseline treemaps which represent the minimal amount of change that must occur in any time-varying treemap for a given change in the data. To verify the performance of the Local Moves algorithms in comparison to existing algorithms we perform an extensive quantitative evaluation of rectangular treemapping algorithms for time-varying data using more than 2000 datasets and 14 state-of-the-art treemapping algorithms.

We identify four representative features of datasets that influence the performance of treemapping algorithms. We use these representative features to propose a novel classification scheme for time-dependent hierarchical datasets. We experimentally test the validity of this classification and analyze the relative performance of the treemapping algorithms across the features on both temporal coherence and visual quality. Finally, we visually summarize the results to aid users in making an informed choice among treemapping algorithms. This chapter is based on joint work with Bettina Speckmann and Kevin Verbeek [126], which appeared in the IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis 2017), and on joint work with Eduardo Vernier, João Comba, Bettina Speckmann, Alexandru Telea, and Kevin Verbeek [149] which appeared in Computer Graphics Forum (Proceedings of Eurovis 2020).

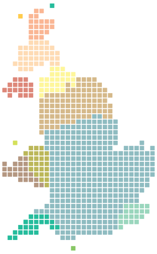
In Chapter 3 we consider uncertain hierarchical data and show how to generate a coherent visualization using treemaps for such data. Using the novel concept of hierarchical uncertainty masks we simultaneously visualize uncertainty and value using area in a treemap. Hierarchical uncertainty masks are based on screen-door transparency to render the uncertainty of each level in the treemap on top of each



other in such a way that the uncertainty for each node is visible simultaneously. This ensures that the intrinsic link between the data and the uncertainty is visible in the visualization, ensuring coherence with respect to uncertainty. Furthermore, we show how to adapt existing treemapping algorithms to support uncertainty masks. To this end, we define a cost function that measures the quality for uncertainty masks to steer and evaluate these algorithms. Finally, we demonstrate the quality of the adapted treemapping algorithms through a computational experiment on real-world datasets. This chapter is based on joint work with Wouter Meulemans, Christoph Schulz, Kevin Verbeek, Daniel Weiskopf, and Bettina Speckmann [125] which appeared in the Proceedings of the 13th IEEE Pacific Visualization Symposium (PacificVis 2020).

## Grid maps

A grid map is an effective and established spatial schematization technique for visualizing geospatial data. Each spatial element in the data, such as a region or a site, is schematized into the same simple tile – often a square, hexagon or other geometry that easily tiles the plane. These tiles are then arranged in such a way as to reflect important characteristics of the spatial dimension often using whitespace to capture salient local features. An example of a grid map visualizing constituencies of the UK is shown on the right.



Aspects such as recognizability and the ability to locate spatial elements based on expected location are important for an effective grid map. An effective grid map must thus be coherent with the underlying spatial dimension to preserve the mental map of the viewer: the tiles maintain properties such as contiguity, neighborhoods, and identifiability of the corresponding spatial elements while the grid map as a whole maintains the global shape of the input.

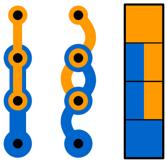
Grid maps schematize the geographic area into simple tiles, and hence deform the spatial dimension. As with any spatial deformation, perfect coherence in the spatial dimension is impossible and thus one must make a trade-off between the different facets of spatial coherence. As a result, computing a coherent grid map is a challenging multi-criteria optimization problem. However, the state-of-the-art [86] shows that simple cases such as close-to-uniform spatial distributions or global shapes with few characteristic features can be solved well using simple tile selection and assignment techniques (as long as sufficient care is taken to guarantee that connected input stays connected in the grid map). For more complex cases, however, current techniques do not maintain spatial coherence well.

**Contribution** In Chapter 4 we introduce a simple fully-automated 3-step pipeline that can compute coherent grid maps for complex data. We observe that we can partition the input shape such that each piece has few characteristic features, and hence becomes a simple subproblem. The coherent solutions for these simple subproblems can then be combined into a coherent overall solution. Each step of our pipeline is a well-studied problem: shape decomposition based on salient features, tile-based Mosaic Cartograms, and point-set matching. Our pipeline is a seamless composition of existing techniques for these problems and results in high-quality grid maps. We provide an implementation, demonstrate the efficacy of our approach on various complex datasets, and compare it to the state-of-the-art. This chapter is based on joint work with Wouter Meulemans and Bettina Speckmann[87] to appear in the IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis 2020) which has received a honorable mention for the best paper award.

## Disjoint colored polygons

Set data can be represented in many different ways. In this thesis we focus on representing spatial set data using hypergraphs with fixed node position. Often, many different drawings are possible for the same graph, but not all are equally suitable for communicating the structure of the graph [99]. One often used criteria to determine the quality of a drawing is how many edges intersect each other. Intersections make it harder to understand the structure of the graph, thereby reducing the effectiveness of the visualization [98].

We consider a hypergraph representation where each hyperedge is represented by a colored polygon. Usually the nodes of the graph are represented by points. This necessarily means that when two hyperedges share a node their associated polygons must overlap. As argued above, this may result in nesting which is a strong visual cue of containment (see right: first figure). Nesting can thus result in a less coherent visualization, as the relations depicted in the visualization and the relations in the data may not match. Alternative representations such as connecting elements using a single smooth curve as proposed by LineSets [1] (see right: second figure) may avoid nesting the hyperedges, but overlap still



occurs at the node. Here, we suggest an alternative representation while still using a polygon to represent each hyperedge (see previous page: third figure). For this representation, we turn each node into a rectangular region: a hyperedge contains a node if its polygon overlaps this rectangular region. Note that multiple hyperedges can overlap a node without crossing each other.

**Contribution** In Chapter 5 we investigate the properties of visualizing spatial set data using the above-described representation using disjoint colored polygons. We assume that the nodes of the sets are positioned on a grid. Each set is represented by a single connected colored polygon and each polygon overlaps exactly those cells that correspond to the data items in the set. As a first exploration of the feasibility of this visual representation we focus on the case where there are only two sets. We derive a necessary and sufficient condition to efficiently recognize whether we can visualize both sets with a single connected polygon. We additionally show that, if both sets can be visualized with a single connected polgyon, then the visual complexity of the polygon in each region is bounded by a small constant. This chapter is based on joint work with Arthur van Goethem, Irina Kostitsyna, Marc van Kreveld, Wouter Meulemans and Jules Wulms [47] which has appeared in the Proceedings of the International Symposium on Graph Drawing and Network Visualization (GD 2017).

**Other contributions** Beside the work included in this thesis, the author also worked on: analyzing the algorithmic complexity of puzzles in the game *The witness* [71]; using testimonial network analysis and visualization to analyze the spread of knowledge in online networks such as Twitter [130]; modeling the epistemic position of an individual in a social network setting [131]; and computing time-varying coherent Demer cartograms [124].

# Chapter 2

# Time-varying Treemaps

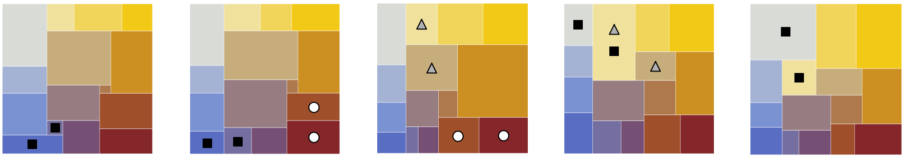


**Figure 2.1** A rectangular treemap over time. In each image the weights of the underlying data have changed. To maintain a balance between aspect ratio and stability we modify the treemap via local moves. Symbols (squares, circles, and triangles) mark the pairs of rectangles to which local moves are applied.

Treemaps are a well-established method for visualizing large hierarchical datasets. Given an input tree whose leaves have several attributes, treemaps recursively partition a 2D spatial region into cells whose visual attributes (area, color, shading, or annotation) encode the tree's data attributes. Most treemaps use rectangles, although there are alternative models such as Voronoi treemaps [7], orthoconvex and L-shaped treemaps [10], and Jigsaw treemaps [152]. In this chapter we focus exclusively on rectangular treemaps, which recursively partition a rectangle into subrectangles.

The *visual quality* of a rectangular treemap is usually measured via the aspect ratio of its rectangles. Kong *et al.* [70] show that large aspect ratios are detrimental to the area assessment, and should thus be avoided. Lu *et al.* [77] argue that the optimal aspect ratio for treemaps should, in fact, be the golden ratio. The aspect ratios of the rectangles in a treemap can become arbitrarily bad: consider a treemap that consists of only two rectangles. If the area of one of these rectangles tends towards zero then its aspect ratio tends towards infinity. Nagamochi and Abe [91] describe an algorithm which computes, for a given set of values and a hierarchy, a treemap which provably approximates the optimal aspect ratio. De Berg *et al.* [10] prove that minimizing the aspect ratio for rectangular treemaps is strongly NP-complete.

Nowadays, large hierarchical datasets are also available over time. Hence, there is a need for *time-varying* treemaps which display changing trees and data values. Ideally, such time-varying treemaps enable the user to easily follow structural changes in the tree and in the data by ensuring coherence with the temporal dimension. In addition to visual quality, time-varying treemaps thus need a quality criterion for temporal coherence which we refer to as stability. Ideally, small changes in the data should result only in small changes in the treemap, that is, data change and layout change should correlate positively. Such stable behavior ensures that the only changes the user sees are due to the data, and not due to the decisions the algorithm makes.

**Definitions and notation** To describe our contribution in greater detail we first introduce some definitions and notation. The input for a rectangular treemapping algorithm is a rectangle $R$ and a set of non-negative values $\mathcal{A} = \{a_1, \ldots, a_n\}$ together with a hierarchy on these values (represented by a tree). We assume the input values are *normalized*, that is, the sum $A = \sum_i a_i$ corresponds to the area of $R$. The output is a treemap $T$, which is a recursive partition of $R$ into a set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of interior-disjoint rectangles, where ($a$) each rectangle $R_i$ has area $a_i$, and ($b$) the regions of the children of an interior node of the hierarchy form a rectangle (associated with their parent). We denote the width and height of a rectangle $R_i$ with $w(R_i)$ and $h(R_i)$. Such a partition of a rectangle into a set of disjoint rectangles is also called a *rectangular layout* or *layout* for short. For a layout $L$, a *maximal segment* is a maximal contiguous horizontal or vertical line segment contained in the union of the boundaries of rectangles in $\mathcal{R}$. We distinguish between two types of layouts: *sliceable layouts* and *non-sliceable layouts*. A layout is sliceable if it can be recursively sliced into two parts along a maximal segment until the layout consists of only a single rectangle. Otherwise the layout is non-sliceable (see Figure 2.2 for an example).

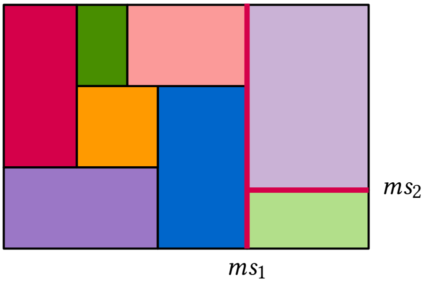


**Figure 2.2** A non-sliceable layout. After slicing along maximal segments $ms_1$ and $ms_2$, the layout can no longer be sliced into two parts along a maximal segment.

In the time-varying setting the input values change over time and become functions $a_i : [0, X] \longrightarrow \mathbb{R}_{\geq 0}$ for each $i$, where the discrete domain $[0, X]$ represents the different time steps in the data. We assume that the hierarchy on the values and $R$ are not time-varying, and that the values $a_i$ are normalized for each time step separately. We use the special value $a_i(t) = 0$ to represent that data element $i$ is not present at time step $t$; and we speak of insertions or deletions if $a_i(t)$ starts or stops to be nonzero, respectively.

Finally, we distinguish *single-level* treemaps and *multi-level* treemaps. A single level treemap has no hierarchy, its tree consist only of a root node with $n$ leaves. Multi-level treemaps correspond to trees with interior nodes in addition to the root. From a theoretical point of view it is sufficient to study single-level treemaps, since all results directly extend to multi-level treemaps; a multi-level treemap can be viewed as multiple nested single level treemaps where the input rectangle of the child is the rectangle determined by the parent. This has the added advantage of removing unnecessary complexity from arguments. Hence, we mostly consider single-level treemaps for the theoretical parts of this chapter.

**Contribution** We present the Local Moves algorithm, a novel stable treemapping algorithm that has high visual quality. In contrast to existing algorithms which produce exclusively *sliceable* layouts, our algorithm can explore the complete space of possible treemap layouts, including *non-sliceable* layouts. We prove that non-sliceable layouts can result in treemaps with higher visual quality.

To verify the efficacy of the Local Moves algorithm as well as existing treemapping algorithms, we perform an extensive quantitative evaluation of rectangular treemapping algorithms for time-varying hierarchical data. To this end, we first propose a new method to measure the stability of time-varying treemaps which explicitly considers the input data. Previously proposed stability metrics measure only the change in the layout and conclude that small changes in the layout are a sign of a stable algorithm. However, to properly measure stability we also need to capture the data change and then correlate the change in the data with the change in the layout. Here, we have to overcome the difficulty that the data and the layout space are a priori incomparable. We solve this problem by introducing the concept of a *baseline treemap* $T^*$ which represents the minimum amount of change that any time-varying treemap must incur (given the change in the data) when moving from treemap $T$ to the next treemap $T'$.

We observe that the performance of treemapping algorithms depend on the characteristics of the datasets used. In order to compare and evaluate treemapping algorithms on a variety of datasets, we therefore propose a novel classification scheme for time-varying datasets. For this classification scheme, we identify four potential representative features that characterize time-varying hierarchical datasets, and classify all datasets used in our experiments accordingly. We experimentally test the validity of this classification on 2405 datasets. Generally, we conclude that the proposed representative features do indeed have predictive value both with respect to visual quality and stability.

Finally, we report on a quantitative evaluation of 14 state-of-the-art rectangular treemapping algorithms on 2405 datasets. We analyze the relative performance of treemapping algorithms across the representative features. Our results are then visually summarized with respect to both visual quality and stability to aid researchers and practitioners in making an informed choice among treemapping algorithms. All datasets, metrics, and algorithms are openly available [137].

**Organization** In Section 2.1 we give an overview of existing treemapping algorithms, and identify representative features that influence their performance. In Section 2.2 we explain the notion of order-equivalence between layouts which we use to generate treemaps with the correct areas in our algorithm. Additionally, we prove that non-sliceable layouts can result in better visual quality than sliceable layouts. In Section 2.3 we present the Local Moves algorithm in detail. We describe the local moves and show that using these moves the complete space of layouts is reachable. We then show how we use these moves in our algorithm. In Section 2.4 we give an overview of existing treemap metrics, and present a new method to measure stability using baseline treemaps. In Section 2.5 we explain our classification scheme for time-varying hierarchical data, and present the data used in our quantitative evaluation. In Section 2.6 we then report on our quantitative evaluation. We first verify the validity of our classification scheme before studying the performance of treemapping algorithms across the representative features. Finally, we give a visual summary of the performance of all treemapping algorithms per data class.

## 2.1 Rectangular treemaps

For a fair comparison during our experiments, we require that treemap rectangles have exactly the correct areas and partition the input rectangle. Algorithms that do not satisfy these requirements are not included in our experimental evaluation. The children of a node in the hierarchy of the treemap are given in a particular order in the input which might reflect meaningful relation between the data. We distinguish two classes of treemaps which either do or do not use this order. For time-varying data we also distinguish between state-aware and stateless treemaps. Contrary to stateless treemaps, state-aware treemaps are not computed separately for each time step, but the layout of the previous time step can be used to compute a new layout. Most treemapping algorithms generate stateless treemaps; we discuss the state-aware treemapping algorithms separately. For each class of algorithms, we identify relevant features that can determine their performance.

**Unordered treemaps** do not (need to) adhere to the input nodes' order when computing the layout. Typically, input weights are sorted to help the algorithm achieve good visual quality. Unordered treemaps in our evaluation include Squarified treemaps (**SQR**) [23] and Approximation treemaps (**APP**) [91]. APP comes with a guaranteed upper bound on the worst-case aspect ratio, while SQR often achieves near-optimal aspect ratios in practice. The visual quality of unordered treemaps is relatively unaffected by high *weight variance*, as reordering weights allows the layout to group similar-size rectangles in the treemap, typically leading to better aspect ratios. Yet, the sorted order of the weights may change rapidly over time, especially if the *weights change* much over time or if the *weight variance* is low. This can negatively affect the stability of the treemaps.

**Ordered treemaps** are required to adhere to the order of nodes as given in the input which roughly ensures that rectangles close to each other in the input are close to each other in the resulting treemap. This typically improves the stability of treemaps, but may reduce visual quality. We include nine ordered treemaps in our evaluation. The first ordered treemaps [113] include the Pivot-by-Middle (**PBM**), Pivot-by-Size (**PBZ**), and Pivot-By-Split (**PBS**) algorithms. Similar algorithms are the Strip algorithm (**STR**) [9] and the Split algorithm (**SPL**) [37]. Other algorithms, like the Spiral algorithm (**SPI**) [139], and the Hilbert (**HIL**) and Moore (**MOO**) algorithms [133], lay out rectangles following a space-filling curve. Finally, the very first treemapping algorithm (Slice-and-Dice (**SND**)) by Shneiderman [112] can also be considered an ordered treemap. While not ordered by design, the resulting (combinatorial) layout depends only on the hierarchy and not on weights. In fact, SND uses the depth in the hierarchy to compute the layout (slicing vertically on even depth and horizontally on odd depth) rather than simply applying the same algorithm recursively. Hence, SND's visual quality strongly depends on the *number of levels* in the input hierarchy. Typically, laying out large rectangles near small rectangles leads to poor aspect ratios. Hence, the visual quality of ordered treemaps is negatively affected by high *weight variance*. However, ordered treemaps are relatively stable over time compared to unordered treemaps, as order is maintained. Finally, *insertions and deletions* may affect the visual quality and stability of ordered treemaps to varying degrees depending on how they are handled exactly.

**State-aware treemaps** can use the layout of the previous time step to compute a new layout and so can largely control their stability. In Section 2.3 we describe our state-aware Local Moves algorithm (**LM**) in detail. The LM algorithm is initialized with a layout generated by the APP algorithm. For each time step it updates the layout with the new areas using the hill-climbing algorithm explained in Section 2.2. This maintains the combinatorial structure up to order-equivalence (see Section 2.2) and hence ensures temporal coherence. If the visual quality deteriorates, the algorithm improves the layout via local moves: small local changes to the structure of the layout. We observed that using up to 4 local moves per time step (**LM4**) allows a good trade-off between stability, visual quality, and efficiency.

To observe the effect of local moves on the visual quality and stability, we also consider a variant of our algorithm where no local moves are performed (**LM0**). The LM0 algorithm thus does not change the structure of the initial layout generated by the APP algorithm except for changes due to insertions and deletions. As such, LM0 is essentially identical to the Greedy Insertion Treemapping (**GIT**) algorithm proposed by Vernier *et al.* [147] which initializes and maintains an SQR layout. Note that SND has a fixed layout if the input hierarchy does not change and is hence very stable, but it does not explicitly use the previous state.

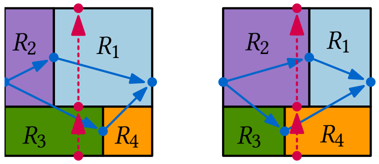
By design, the stability of state-aware treemaps is relatively unaffected by frequent *weight changes* over time. Also, the visual quality is initially high as a stateless treemapping algorithm with high visual quality is used to generate the initial treemap. However, since layouts cannot change much over time, the visual quality of state-aware treemaps generally decreases if *weights change* significantly. Any state-aware treemapping algorithm needs to handle insertions and deletions explicitly. On the one hand, frequent *insertions and deletions* may cause the visual quality to deteriorate as treemaps are never recomputed completely. On the other hand, insertions can be used to locally improve the visual quality. GIT strongly relies on insertions, as they are the only option it has to improve its layout over time. This is in contrast to LM4 which can improve its layout using both local moves and insertions.

Finally, note that the *number of levels* in the input hierarchy can have a strong effect on all classes of algorithms. In general, more levels imply less freedom in the layout strategy. As a result, unordered treemaps become more similar to ordered treemaps. Overall, the visual quality tends to decrease and the stability tends to increase.

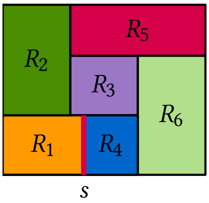
## 2.2 Layouts

For a given set of areas $\mathcal{A}$ there are multiple ways to draw a treemap, that is, there are multiple layouts that can represent the same set of areas. We want to find the layout that has the highest visual quality. To do so, we need to explore the space of possible layouts. All current rectangular treemapping algorithms produce only sliceable layouts. This is obvious for SND since slicing cuts are an integral part of the algorithm. The APP, GIT, SQR, SPI, SPL, STR, and PB{M,Z,S} algorithms all explicitly construct treemaps using slicing cuts. The treemaps created using space-filling curves (HIL and MOO) are also sliceable: the base of the recursion are four rectangles and every layout with 4 rectangles is sliceable. In this section we prove that the visual quality of treemaps can be improved substantially by considering all possible layouts, sliceable and non-sliceable. Our algorithm, presented in Section 2.3, is the first treemapping algorithm that can produce all possible layouts.

Two layouts representing different areas cannot be the same. Nonetheless, they can have a very similar structure. We therefore consider a combinatorial equivalence between layouts. We define a partial order on the maximal segments in a layout $L$ of the same orientation as follows. For two horizontal maximal segments $s_1$ and $s_2$ we say that $s_1 < s_2$ if $s_1$ is below $s_2$, and there exists a rectangle $R_i$ that spans from $s_1$ to $s_2$. Vertical maximal segments similarly define a partial order from left to right. Following Eppstein *et al.* [40], we say that two layouts $L$ and $L'$ are *order-equivalent* if the partial orders for $L$ and $L'$ are isomorphic. An example of order-equivalent layouts is given in the figure on the right, blue and red arrows indicate the partial order on the vertical and horizontal maximal segments respectively. In [40] it was shown that, for any layout $L$, there is always exactly one layout $L'$ that is order-equivalent to $L$ and correctly represents a given set of areas. Thus, for any fixed set of areas $\mathcal{A}$, the possible ways to draw a treemap with areas $\mathcal{A}$ corresponds to the set of order-equivalence classes of all possible layouts.



**Sliceable and non-sliceable layouts** If a layout $L$ is sliceable, then all layouts order-equivalent to $L$ are also sliceable. Existing rectangular treemapping algorithms hence exclude a large number of options from consideration which may result in treemaps of sub-optimal visual quality. We aim to show this formally. Below we prove that, for certain sets of areas, the maximum aspect ratio of any sliceable layout is much larger than the maximum aspect ratio of the optimal layout.



We say that a rectangle $R_i \in \mathcal{R}$ is *grounded* if $R_i$ is bounded by at least one maximal segment $s$ for which it is the only rectangle on that side of $s$. We claim that in a sliceable layout all rectangles are grounded. Indeed, if this is not the case then there must be a rectangle $R_i$ such that all four bounding maximal segments have at least two rectangles on the side of $R_i$. This results in a "windmill pattern" with $R_i$ in the center ($R_3$ on the right). It is not hard to see that any layout containing a "windmill pattern" is non-sliceable. We can now prove the following theorem.

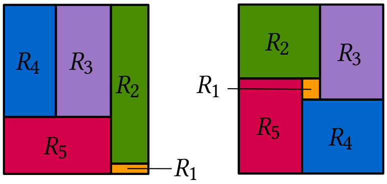
**2.2.1 Theorem.** *The maximum aspect ratio of a sliceable layout L is at least* $\sqrt{a_j/a_i}$, *where* $a_i$ *and* $a_j$ *are the smallest and second-smallest area in the layout, respectively.*

*Proof.* Let $\rho$ be the maximum aspect ratio of $L$ and let $R_i$ be the rectangle with the smallest area in $L$. Let rectangle $R_j$ be adjacent to $R_i$ such that $R_i$ is grounded in the maximal segment shared with $R_j$. Without loss of generality we assume that rectangle $R_j$ lies to the right of rectangle $R_i$. From the grounded property we get that $h(R_i) \geq h(R_j)$. This also implies that $a_j/a_i \leq w(R_j)/w(R_i)$. From the definition of $\rho$ we get that $h(R_i) \leq \rho w(R_i)$. We further get that

$$\rho \geq \frac{w(R_j)}{h(R_j)} \geq \frac{w(R_j)}{h(R_i)} \geq \frac{w(R_j)}{\rho w(R_i)}.$$

As a result, $\rho^2 \geq w(R_j)/w(R_i) \geq a_j/a_i$. Thus the maximum aspect ratio of $L$ is at least $\sqrt{a_j/a_i}$. This is minimized when $R_j$ has the second smallest area in $L$. □

Consider the following concrete example with 5 areas: $a_1 = 1$ and $a_2, a_3, a_4, a_5 = 16$. According to Theorem 2.2.1 any sliceable layout will have a maximum aspect ratio of at least $\sqrt{16/1} = 4$. However, there exists a non-sliceable layout with these areas that has a maximum aspect ratio $\approx 1.333$. In fact, this difference in maximum aspect ratio can be arbitrarily large: as $a_1$ tends to 0, the maximum aspect ratio of the non-sliceable layout tends to 1 while the maximum aspect ratio of any sliceable layout tends to $\infty$.



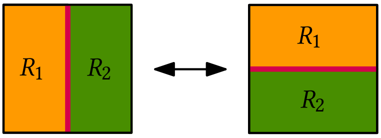
## 2.3 Local moves algorithm

Our stable treemapping algorithm uses the concept of *local moves*. A local move changes the order-equivalence class of the layout $L$ by changing the layout $L$ locally. Local moves allow us to traverse between all order-equivalence classes of layouts. Intuitively, we can keep a treemap stable over time by limiting the number of local moves between any two time steps. At the same time, the more local moves we allow the better the visual quality can be. Local moves hence give us the power to control the trade-off between stability and visual quality. A local move typically changes the areas of the involved rectangles. We can correct the areas in the resulting layout $L'$ using the hill-climbing algorithm by Eppstein *et al.* [40] (for details see Section 2.3.2).

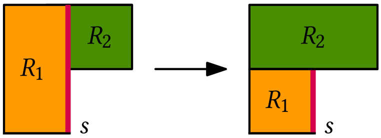
The final layout is order-equivalent to $L'$. Note that two order-equivalent layouts may have different adjacencies across maximal segments. However, since these changes occur only along maximal segments, they influence the relative positions of rectangles only mildly. We hence claim that the algorithm is stable if we allow only a small number of local moves. The experimental evaluation in Section 2.6 supports this claim.

Our local moves are inspired by the work of Young *et al.* [160] which uses a representation of rectangular layouts with twin binary trees. The authors show how to use these twin binary trees and an additional labeling to transform any two rectangular layouts into each other. Their particular labeling is not suitable for the context of treemaps and moreover the representation by twin binary trees is somewhat cumbersome. Below we hence introduce two new local moves which operate directly on the treemap: *stretch moves* and *flip moves*. We prove that one can transform any two order-equivalence classes of layouts into each other using only these two moves.

**Flip move** Let $R_1$ and $R_2$ be two rectangles that partition a larger rectangle. A flip move flips the adjacency between $R_1$ and $R_2$ from horizontal to vertical or vice versa inside this larger rectangle.



**Stretch move** Let $s$ be a maximal segment and let $R_1$ and $R_2$ be two rectangles adjacent to one of the endpoints of this segment. Without loss of generality we assume that $s$ is a vertical maximal segment. If rectangles $R_1$ and $R_2$ do

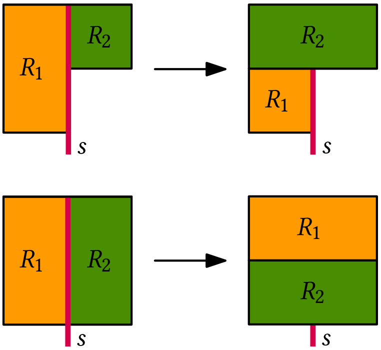


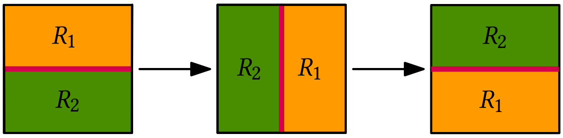
not have the same height we can apply a stretch move. Let rectangle $R_2$ denote the rectangle with the smallest height. To apply the stretch move we then stretch rectangle $R_2$ over rectangle $R_1$.

### 2.3.1 Transforming rectangular layouts using local moves

We now prove that we can transform any layout $L$ into any other layout $L'$ using only local moves. For this transformation we need the notion of a *vertical stack layout*. A layout is a vertical stack layout if it has only horizontal (inner) maximal segments. The transformation from $L$ to $L'$ can now be summarized as follows. First, we transform $L$ into a vertical stack layout. Next, we transform this vertical stack layout into another vertical stack layout. Finally, we transform the resulting vertical stack layout into $L'$. To show the existence of this transformation we need the three following components.

**Transforming a layout to a vertical stack layout** To transform a layout $L$ to a vertical stack layout we need to eliminate all vertical (inner) maximal segments. Let $s$ be a vertical maximal segment of $L$. Furthermore, let $R_1$ and $R_2$ be the rectangles adjacent to the left top and right top of $s$ respectively. Now first assume that $R_1$ and $R_2$ do not have the same height, and assume without loss of generality that the height of $R_2$ is smaller than the height of $R_1$. In this case we use a stretch move to stretch $R_2$ over $R_1$. If $R_1$ and $R_2$ have the same height then we can use a flip move on $R_1$ and $R_2$. Note that, in both cases, we reduce the number of rectangles adjacent to $s$ by at least one. When there are no more rectangles adjacent to $s$, $s$ will cease to exist. Furthermore, our operations do not introduce new vertical maximal segments. We can thus repeatedly apply this procedure until all vertical maximal segments have been eliminated.





**Transforming vertical stack layouts** Consider any two adjacent rectangles $R_1$ and $R_2$ in a vertical stack layout. We can swap $R_1$ and $R_2$ in the vertical stack order by applying two flip moves to $R_1$ and $R_2$. Since we can swap any two adjacent rectangles, we can produce any order of rectangles in the vertical stack layout (this process is the same as sorting with *BubbleSort*).

**Inverting local moves** It is easy to see that all local moves can be inverted. Trivially, a flip move is its own inverse. A stretch move that stretches $R_2$ over $R_1$ can be inverted by a stretch move that stretches $R_1$ over $R_2$.

We can now prove the following theorem:

**2.3.1 Theorem.** *For any two layouts $L_1$ and $L_2$ with the same set of rectangles, we can transform $L_1$ into $L_2$ using only stretch moves and flip moves.*

*Proof.* We can transform $L_1$ into a vertical stack layout $L_1'$ as described above. Similarly, we can transform $L_2$ into a vertical stack layout $L_2'$. To transform $L_1$ into $L_2$, we first transform $L_1$ into $L_1'$. Next, we transform $L_1'$ into $L_2'$ using appropriately chosen swaps of adjacent rectangles. Finally, we transform $L_2'$ into $L_2$ by inverting the local moves used to transform $L_2$ into $L_2'$. □

We can additionally show that if the number of rectangles in $L_1$ and $L_2$ is $n$, then we need at most $O(n^2)$ local moves to transform $L_1$ into $L_2$.

Note that the proof above is a so-called "constructive proof of existence". Our argument ($i$) shows that there always is a set of local moves to transform one layout into the other, and ($ii$) it describes a way to find these moves. Clearly, the resulting transformation is not very natural and we do not intend to use this transformation. Now that we have proven that a transformation always exists, we can find more suitable transformations in practice.

### 2.3.2 Algorithm

We now describe our stable treemapping algorithm *Local Moves* for time-varying hierarchical data. Our algorithm uses the previous treemap to generate the next one. We therefore need to describe how to transform a treemap $T(t)$ with areas $\mathcal{A}(t) = \{a_1(t), \dots, a_n(t)\}$ into a treemap $T(t+1)$ with areas $\mathcal{A}(t+1) = \{a_1(t), \dots, a_n(t)\}$. To simplify notation we denote $T(t)$,$\mathcal{A}(t)$,$T(t+1)$,$\mathcal{A}(t+1)$ by $T$,$A$,$T'$ and $A'$ respectively.

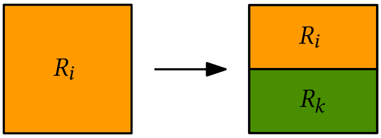
We first consider only a single-level treemap $T$. We construct the initial treemap using the algorithm (APP) by Nagamochi and Abe [91]. To transform $T$ into $T'$ we use a very simple approach. First, we update the treemap $T$ to have the areas in $A'$ using the hill-climbing algorithm by Eppstein *et al.* [40].

The idea of the algorithm by Eppstein *et al.* is as follows. As shown in [40] there is an induced bijection between the space of coordinates of the maximal segments (*segment space*) and the space of the areas of the rectangles (*area space*). Hence, given a (tangent) vector in the area space, in particular $A' - A$, we can compute the corresponding tangent vector $x$ in the segment space by solving the linear equation $Jx = A' - A$ where $J$ is the Jacobian matrix of the bijection. Thus, we can locally change the areas from $A$ to $A'$ by moving the maximal segments in the direction of $x$. The Jacobian matrix $J$ is sparse and can easily be computed as, for each rectangle, the area simply depends on the coordinates of the 4 maximal segments bounding the rectangle. We can now proceed as in a gradient descent approach by iteratively changing the maximal segment coordinates by $\varepsilon x$ for small enough $\varepsilon$, until we obtain the areas in $A'$.

Next, we attempt to improve the visual quality of the layout by applying up to $d$ local moves, where $d$ is some predefined small constant (in our experiments $d = 4$). A naive approach would simply try all possible sets of at most $d$ local moves. In Section 2.3.3 we explain how to choose a suitable subset of possible moves to optimize performance. The areas of the resulting layouts (after at most $d$ local moves) are then again adjusted using the hill-climbing algorithm by Eppstein *et al.* [40] to generate an order-equivalent layout with the correct areas. We use the layout with the best average aspect ratio to construct $T'$. Note that we do not change the layout if doing so would lead only to a minor improvement in aspect ratios. Therefore, if $L$ is the layout of $T$ with updated areas $A'$, then we change $L$ into $L'$ only if the sum of aspect ratios in $L'$ is at least some predefined constant $c$ lower than the sum of aspect ratios in $L$.

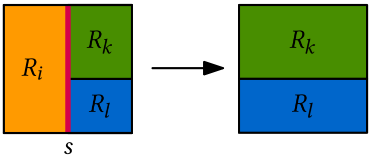
If $T$ is a multi-level treemap, then we use our algorithm recursively on the rectangles that represent subtrees. That is, we first transform the single-level treemap which is formed by the root of $T$ and its children using a set of local moves. Then we recursively apply the algorithm inside each of the resulting rectangles $R_i$. The choice of moves is restricted to those moves that involve only subrectangles of $R_i$ which ensures that the hierarchy information is maintained. Clearly, changing the layout on a higher level of $T$ has more impact than changing it on a lower level, as it affects all subtreemaps of $T$. We account for this by adapting the value of $c$ according to the height of the level. In our implementation we use $c = 4 * \sqrt{\text{height of the level}}$.

**Handling insertions** When additional data becomes available we need to insert new rectangles in the treemap. We insert such new rectangles before performing



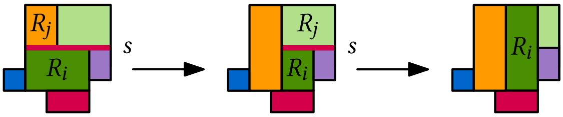
any local moves. To add a new rectangle $R_k$ to the treemap, we partition an existing rectangle $R_i$ into two subrectangles $R_i$ and $R_k$ . We choose $R_i$ in such a way that the maximum aspect ratio is minimized. When multiple rectangles need to be inserted we insert them in order of largest area, as rectangles with large areas have more impact on the layout. However, when the number of rectangles that need to be inserted is large, inserting them sequentially is undesirable. It is both computationally expensive, as the layout needs to be recalculated after every insertion, and can result in low visual quality, as each insertion is optimized by itself and not as a whole. Therefore, when the number of insertions below a node $v$ in the hierarchy is larger than the number of children of $v$ we regenerate the layout locally for $v$ using the APP algorithm. This approach allows us to handle volatile datasets where items appear and disappear rapidly.

**Handling deletions** For similar reasons we may need to delete a rectangle $R_i$ from the treemap. Deleting rectangles is slightly more involved than inserting rectangles, and happens before any local moves are performed, but after new rectangles have been inserted. There are two cases we need to consider when deleting $R_i$:



***$R_i$* is grounded:** There necessarily exists a maximal segment $s$ for which $R_i$ is the only rectangle on one side of $s$. To delete $R_i$ we stretch all rectangles on the other side of $s$ over $R_i$ using stretch moves.

***$R_i$* is not grounded:** $R_i$ must be in the center of a windmill pattern. We apply stretch moves to $R_i$ until $R_i$ becomes grounded and we are in the first case. Let $e$ be the edge of $R_i$ that is adjacent to the fewest rectangles on the other side. Since $R_i$ is in the center of a windmill pattern, $e$ must include an endpoint of a maximal segment $s$. Without loss of generality $e$ is above $R_i$ and the endpoint of $s$ is on the left side of $e$. Then, as long as there is more than one rectangle on the top side of $e$, we stretch the leftmost of those rectangles $R_j$ over $R_i$. Once there is only one rectangle $R_j$ on the top side of $e$ we stretch $R_i$ over $R_j$. It is possible that $R_i$ is still not grounded, but then it is now part of a larger windmill pattern. In that case we repeat the procedure above until $R_i$ finally becomes grounded.



We summarize our algorithm in the following pseudocode where $f(L)$ measures the sum of aspect ratios in a layout $L$ and correctAreas($L, A'$) is an implementation of the algorithm by Eppstein *et al.* [40].

**Algorithm 1** LocalMoves($T$,$A'$,$d$)

```
$c = 4 * \sqrt{height(T)}$
if $T$ is empty then
    Generate $T'$ using the Approximation algorithm.
else
    $L$ = correctAreas($L, A'$)
    Insert rectangles to $T$ that need to be in $T'$.
    Delete rectangles from $T$ that are not in $T'$.
$Q_0 = \{L\}$
$L_{best} = L$
for $i = 1$ to $d$ do
    for $L' \in Q_{i-1}$ do
        for all possible local moves $m$ on $L'$ do
            $L''$ = apply($m$, $L'$)
            $L''$ = correctAreas($L''$, $A'$)
            if $f(L'') < f(L_{best})$ then
                $L_{best} = L''$
            $Q_i = Q_i \cup \{L''\}$
if $f(L_{best}) < f(L) - c$ then
    Let the layout of $T'$ be $L_{best}$
for all children $T_c$ of $T$ do
    LocalMoves($T_c$, $A'(T_c)$, $d$, $c$)
```

## 2.3.3 Improving performance

The naive algorithm described above is not very efficient for two reasons: ($i$) the number of layouts considered by the algorithm is exponential in $d$, and ($ii$) updating the areas using the hill-climbing algorithm is not very efficient. We address these two issues below.

**Reducing the number of layouts** We first compute all layouts that are the result of applying one local move. Of these layouts, we keep only the layouts which improve the aspect ratio. Of these layouts, we keep only the $k$ layouts with the smallest

**Algorithm 2** correctAreas($L$,$A'$)

```
Let $A$ be the areas in $L$.
while $\|A - A'\|$ is not small enough do
    Let $J$ be the Jacobian matrix of mapping segments to areas.
    Solve $Jx = A' - A$ for $x$.
    Move maximal segments of $L$ by $\varepsilon x$.
    Recompute areas $A$ of $L$.
return $L$
```

aspect ratios (in our implementation we use $k = 4$). When applying a subsequent local move to one of the remaining layouts, we consider only those local moves that involve a maximal segment for which the adjacencies have been changed by the local move. Afterwards, we again keep only the layouts which improve the aspect ratio. We repeat the procedure until we have applied $d$ local moves per layout.

Although this approach may not find the best possible layout, it does perform well in practice and the number of layouts considered is no longer exponential in $d$.

**Updating areas more efficiently** Computing the correct areas for general (possibly non-sliceable) layouts is significantly more difficult than computing the correct areas for sliceable layouts (which can simply be computed recursively). However, most layouts contain large components that are sliceable. We can use this fact to speed up our algorithm. While we can find a maximal segment $s$ that slices the layout, we simply place $s$ according to the areas of the rectangles on its two sides, and continue recursively on both sides of $s$. When the layout is not sliceable, we try to find maximal segments that have a single rectangle on both sides. These maximal segments can be removed and reinserted later as a slicing maximal segment. Finally, when no such maximal segments remain, we use the hill-climbing algorithm described in Section 2.3.2 to position the remaining maximal segments. This approach speeds up our algorithm substantially in practice.

## 2.4 Metrics

In order to perform a quantitative evaluation for treemapping algorithms, we require metrics to determine the quality of a treemap. Wattenberg [152] identifies several desirable properties of treemaps: (1) nicely shaped regions (visual quality), (2) stability with regard to changing leaf values, (3) stability with regard to changing tree structure, and (4) preservation of order information. Regarding Property (3), the tree structure can change in various ways: for example, nodes can merge or split, nodes can change parents, or there are general insertions and deletions. In our experiments we do not make any assumptions on the type of changes to the tree structure, and hence treat them as general insertions and deletions. Furthermore, we do not assume that the order of the values in the data is meaningful in general. Thus, we consider the following two important criteria to evaluate treemaps: *visual quality* and *stability*. We discuss well-established metrics for both below and also introduce a method that allows any existing stability measure for time-varying treemaps to explicitly take data change into account. We compute metrics for each leaf rectangle separately and then aggregate these values for each algorithm and dataset (see Section 2.6 and [137] for details). Note that we do not compute metrics for non-leaf nodes.

### 2.4.1 Visual quality

The weight information in a treemap is conveyed by the areas of its rectangles. Since areas of rectangles closer to squares are visually easier to estimate than areas of elongated rectangles, the visual quality of a treemap is commonly measured by the aspect ratio of its rectangles. Although it has been proposed that the ratio should be close to the golden ratio [77] instead of the minimum aspect ratio of 1, it is commonly accepted that strongly elongated rectangles hinder readability of treemaps. We thus aim for the overall goal of making rectangles as square as possible, or similarly, minimizing the number of elongated rectangles. For a rectangle $R_i$ of width $w(R_i)$ and height $h(R_i)$, we define the aspect ratio $\rho(R_i)$ as

$$\rho(R_i) = \min(w(R_i), h(R_i)) / \max(w(R_i), h(R_i)). \tag{2.1}$$

Observe that this definition is the inverse of the usual definition for aspect ratio. Its values range from 0 to 1 where values of $\rho$ close to 0 are considered “bad” and values close to 1 are considered “good”. The bounded range allows for easy aggregation. Note that, compared to the usual definition of $1/\rho$, rectangles with larger aspect ratios have a smaller influence on the aggregated score.

### 2.4.2 Stability

Evaluating the stability of a treemap is more involved than evaluating visual quality. Consider treemaps at two consecutive time steps $T(t)$ and $T(t+1)$. To simplify notation, we denote the former and the new treemap by $T$ and $T'$ respectively. We also denote the rectangle areas in $T$ and $T'$ by $\{a_1, \ldots, a_n\}$ and $\{a'_1, \ldots, a'_n\}$, respectively. For a stable treemapping algorithm, the (visual) difference between $T$ and $T'$ should roughly correspond to the difference between $\{a_1, \ldots, a_n\}$ and $\{a'_1, \ldots, a'_n\}$. Note that the combination of large changes in data values and small changes in the layouts is unlikely since rectangle areas in treemaps must exactly match the data values. Hence, we actually want to measure *instability*, that is, large layout changes that are not caused by large data changes.

Most existing treemap stability metrics consider only the visual change in the layout of the treemap $d(T, T')$, usually computed by evaluating the change $\delta(R_i, R'_i)$ for each rectangle separately and aggregating it over all rectangles. The first stability measure for treemaps was proposed by Shneiderman and Wattenberg [113] who define $\delta$ as the Euclidean distance between the vectors $(x(R_i), y(R_i), w(R_i), h(R_i))$ and $(x(R'_i), y(R'_i), w(R'_i), h(R'_i))$ where $x$, $y$, $w$, and $h$ are the coordinates of the top-left corner, width, and height of a rectangle, respectively. They then define $d$ as the average over all rectangles. Hahn *et al.* [54, 55] simplify this metric by defining $\delta$ as the distance moved by the centroid of a rectangle, again defining $d$ as the average. Tak and Cockburn [133] use the same $\delta$ as [113], but define $d$ as the variance over all values computed by $\delta$. They also propose a drift metric which measures how much a rectangle moves away from its average position over a long period. Chen *et al.* [28] propose a metric to quantify the ability of users to track time-varying data in treemaps which is closely related to the aforementioned drift metric. Scheibel *et al.* [107] introduced two layout-change metrics: The *average aspect ratio change* defines $\delta$ as the relative change between the aspect ratios of $R_i$ and $R'_i$, and defines $d$ as the average. The *relative parent change* defines $\delta$ as the relative change of the distance between the center of a rectangle and the center of its parent, again defining $d$ as the average. A different approach measures layout change using pairs of rectangles. Hahn *et al.* [56] introduce the *relative direction change* which measures how much the angle from the center of $R_i$ to the center of $R_j$ changes for every pair of rectangles $R_i$ and $R_j$ . Finally, Sondag *et al.* [126] proposed the *relative position change* which instead measures how much the relative position of $R_i$ with respect to $R_j$ changes for every pair of rectangles $(R_i, R_j)$. The distance $d$ is then defined as the average over all pairs of rectangles.

Summarizing, we distinguish two types of layout-change metrics: (1) *absolute* metrics measure how much individual rectangles move/change, and (2) *relative* metrics measure how much positions of pairs of rectangles change relative to each other. For our experiments, we use both an absolute and a relative metric. In particular, as an absolute metric, we use the *corner-travel distance* which is a well-known metric used in computer vision to quantify change between two shapes using feature points [132, 142]. In the vision community, it was established already many years ago [13, 111] that corners are a perceptually useful feature to identify and track. Besides this perceptual validation, the corner-travel metric lies also within a small bounded factor of the original metric introduced by Shneiderman and Wattenberg [113]. Specifically, let $p_i$, $q_i$, $r_i$, and $s_i$ ($p_i'$, $q_i'$, $r_i'$, and $s_i'$) be the positions of the corners of a rectangle $R_i$ ($R_i'$), and let $R$ be the input rectangle. We define the normalized corner-travel (CT) distance for a rectangle as

$$\delta_{\mathrm{CT}}(R_i, R_i') = \frac{\|p_i - p_i'\|_1 + \|q_i - q_i'\|_1 + \|r_i - r_i'\|_1 + \|s_i - s_i'\|_1}{4\sqrt{w(R)^2 + h(R)^2}}. \tag{2.2}$$

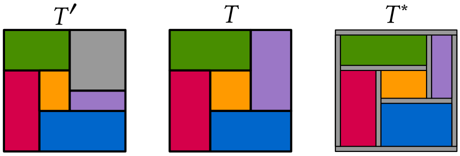
where $\|x\|_1$ denotes the $\ell_1$ norm. Simply put, $\delta_{\mathrm{CT}}$ is the corner-to-corner correspondence distance between $R_i$ and $R_i'$. Note that $0 \leq \delta_{\mathrm{CT}}(R_i, R_i') \leq 1$, since a rectangle corner can travel by at most the length of the diagonal of $R$.

As a relative metric we use the *relative position change* [126]. We established experimentally that the corner-travel and the relative position change metric correlate clearly on more than 2000 datasets. Hence, in Section 2.6 we report only on experiments using the corner-travel distance. All other data can be found online [137].

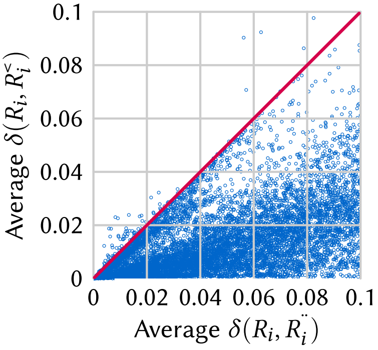
**Data change** The stability metrics discussed above do not take data change into account. If data changes by a large amount, then the layouts should be allowed to change significantly without considering this to be instability. To add data change to a stability metric one can consider the difference or ratio between the layout change and the data change [147, 148]. However, there are two problems: (1) we need a way to measure data change, and (2) the metric spaces for data and layouts need to be comparable. For example, data change can be measured in terms of changes of rectangle *areas* (since these correspond to the data). However, layout changes such as the corner-travel distance measure *lengths*, not areas. Areas and lengths are not directly comparable, and thus their ratios or differences may not be meaningful. Although such metrics could be made comparable by suitable normalization, such adaptations are necessarily metric-specific and ultimately result in numbers whose meaning is not clear.

**Baseline treemap** We overcome the above issues with a new method that captures data change *in the layout space*. To this end, we define a *baseline* treemap $T^*$ with respect to $T$ and $T'$. The layout of $T^*$ is order-equivalent to the layout of $T$. However, the areas of the rectangles in $T^*$ are the areas $\{a_1', \ldots, a_n'\}$ of $T'$. The idea is that $T^*$ aims to minimize the layout distance to $T$ among all treemaps with the areas of $T'$. Put differently: $T^*$ approximates the minimum amount of change that any time-varying treemap must incur when moving from $T$ and its associated area values $\{a_i\}$ to the next treemap $T'$ and its area values $\{a'_i\}$. As a result, $d(T, T^*)$ is a good metric for data change in the layout space.

We construct $T^*$ for each tested algorithm and each time step using a hill-climbing algorithm (see Section 2.2). If rectangles are inserted or deleted $T^*$ cannot be order-equivalent to $T$, so we handle insertions and deletions separately. Dealing with deletions is easy: we simply let the areas go to zero. For insertions, we must be more careful. Indeed, while we consider only rectangles present in both $T$ and $T'$ when measuring stability ($R_i$ and $R'_i$ in Equation 2.2), inserted rectangles can strongly impact the positions of rectangles in $T^*$. We observe that the baseline treemap does not need to be a proper treemap: it only needs to capture how much rectangles must minimally move to update to the new data. To minimize the movement of the rectangles due to insertions (and hence be as stable as possible) we distribute the cumulative area of the inserted rectangles over the “walls” (borders) of treemap $T$ evenly. To do so, we replace every maximal segment $ms$ in $T$ by a rectangle $r_{ms}$, and assign $r_{ms}$ a portion of the inserted area proportional to the length of $ms$. Hence, all walls become equally thick and the original rectangles of $T$ need to move little to yield $T^*$. Note that $T^*$ is not an actual treemap that represents the input data. Instead, $T^*$ it is an artificially created treemap (hence, the name ‘baseline’) which has many additional (gray) rectangles that represent the data change between time steps.



The baseline treemap $T^*$ as proposed here is not a perfect baseline as it does not always minimize the movement of every rectangle. To gain insight into how well $T^*$ captures the minimal movement of every rectangle, we visualize the relation between the average layout change between $T$ and $T'$ or $T^*$ for a random 25% sample of all algorithms and datasets. Nearly all points lie on or below the diagonal and thus there are few cases where any of the 14 treemapping algorithms needs less move-



ment of the rectangles than the baseline. Therefore, the layout change between $T$ and $T^*$ is a good estimate for the minimum necessary layout change between $T$ and $T'$, and thus a good measure for data change.

**Stability metric** We can now define a stability metric that takes data change into account. Consider a rectangle $R_i$ and the corresponding rectangles $R'_i$ and $R^*_i$ in $T'$ and $T^*$, respectively, and let $\delta$ be the layout-change function for single rectangles. Two natural choices for spatial stability are the difference or ratio between $\delta(R_i, R'_i)$ and $\delta(R_i, R^*_i)$. Our experiments showed that the difference is typically more informative, that is, it exhibits clearer, more pronounced patterns, than the ratio. Hence, we define the stability of a single rectangle as

$$\sigma(R_i) = \max(0, \delta(R_i, R'_i) - \delta(R_i, R^*_i)) \tag{2.3}$$

Note that $\sigma(R_i) = 0$ if $\delta(R_i, R'_i) \leq \delta(R_i, R^*_i)$, which is possible. Indeed, a value of 0 for $\sigma(R_i)$ represents “very stable”, and $R^*_i$ is considered to be (roughly) as stable as possible.

**Limitations** The stability metrics we use focus only on consecutive time steps. The stability of time-varying treemaps could conceivably be influenced by effects that span multiple time steps which our metrics do not capture directly. However, we believe that the most salient events influencing stability occur between consecutive time steps and hence we focus on this scenario.

## 2.5 Data

The visual quality and stability of treemaps clearly depend on the datasets used. Simply measuring the average performance over a (large) collection of datasets does not reveal such information. We aim to provide sufficient insight so that both practitioners and researchers can make informed choices about which algorithm to use for their data. For this, we study the performance of treemaps as a function of the characteristics of the input data. We classify the datasets into data classes along with explicit features and evaluate the metrics for all treemapping algorithms per class.

### 2.5.1 Data features

Our methodology is inspired by the framework proposed by Smith-Miles *et al.* [123] to objectively measure the performance of algorithms across datasets. For each dataset, we compute a number of features that (hopefully) capture the characteristics influencing the relative performance of treemapping algorithms. As a result, every dataset is represented by a point in a low-dimensional *feature space* $\mathcal{F}$. Similar feature-based approaches are also used to measure the relative performance of dimensionality-reduction methods [41] or in machine learning [14]. Based on the discussion of treemapping algorithms in Section 2.1 we identify the following four features: **1.** levels of hierarchy, **2.** variance of node weights, **3.** weight change, and **4.** insertions and deletions.

Obviously, other features could be used to characterize time-varying hierarchies such as the minimum, maximum, and average node degrees, the (im)balance of the tree structure [18, 72], or the number of nodes. Two seemingly obvious candidates for features that we do not currently consider are the *number of nodes* and the *branching factor* (i.e., the average internal node degree). Arguably, the number of levels in the hierarchy, the branching factor, and the number of nodes correlate to some degree. For example, if the hierarchy has only one level, then the branching factor and the number of leaves are the same. Hence, we should include at most two of these features in our analysis. Among these three features, the number of levels is certainty a discriminating factor between algorithms, see our discussion in Section 2.1. Furthermore, all algorithms we consider, with the exception of SND, are recursive and treat each level independent from the preceding ones. Hence, one can argue that the branching factor which determines the number of nodes that have to be handled during a single step of this recursion is a more relevant feature than the total number of nodes. Nevertheless, we decided not to include the branching factor in our evaluation for the following two reasons. First, from the description of the algorithms, it seems that the branching factor is likely less relevant for their *relative* performance than the other four chosen features. That is, the descriptions do not give any indication that the branching factor is able to predict if an algorithm $A$ will perform better than an algorithm $B$ on a given dataset. Second, it is very difficult to define meaningful value-ranges for the branching factor and then to find datasets that cover these ranges in combination with all other data features. Given that the number of data classes and, correspondingly, the number of datasets needed for a meaningful evaluation, grows exponentially with the number of features chosen (see Section 2.5.2), we decided to restrict ourselves to four features. While we cannot exclude that the branching factor may influence relative performance, we do believe that the four features chosen have higher predictive value.

### 2.5.2 Data classes

Using the feature space $\mathcal{F}$ we partition all datasets into classes. For each feature we define a small number of subclasses based on only that feature. The data class of a dataset is then defined as the combination of the subclasses for each feature. We determined the value-ranges defining the subclasses by analyzing the distribution of feature values over our 2405 real-world hierarchical datasets.

**Levels of hierarchy** We distinguish between 3 subclasses: 1 level (1L), 2 or 3 levels (2/3L), and more than 3 levels (4+L). Most hierarchical datasets we have analyzed have 2 or 3 levels. This number of levels is quite common for datasets that are visualized via treemaps, since they frequently concern geospatial subdivisions such as countries, continents, and their subregions grouped by a classification scheme such as the World Bank regional classification. Furthermore, visually understanding the node nesting in deeper treemaps becomes difficult [23, 150]. A special case are datasets with only 1 level, that is, sets of weight values. Such datasets are also often visualized by treemaps, as these are more space-filling than alternatives such as bar charts [150]. These datasets are challenging for treemaps that implicitly use the depth of the hierarchy. Finally, we consider datasets with more than 3 levels which correspond to deep hierarchies such as, for example, file systems or software architectures [54, 56, 148].

**Variance of node weights** We distinguish between 2 subclasses: low weight variance (LWV) and high weight variance (HWV). To ensure that the total number of tree nodes does not strongly influence our classification, we use the coefficient of variation $\sigma/\mu$ to determine the subclass. The standard deviation $\sigma$ and the mean $\mu$ are computed over all leaf weights over all time steps. We say that there is low variance if $\sigma/\mu \leq 1$ and high variance if $\sigma/\mu > 1$, respectively.

**Weight change** We distinguish between 3 subclasses: low weight change (LWC), regular weight change (RWC), and spiky weight change (SWC). The weight change of a single rectangle is measured by the absolute difference in the relative area (with respect to the input rectangle R) between consecutive time steps. The weight change of a treemap between two time steps is defined as the sum of weight changes of all rectangles. To determine the subclass of a dataset we use the distribution of weight changes between time steps over all time steps in the dataset, specifically the mean $\mu$ and the standard deviation $\sigma$. Datasets with low weight change have $\mu < 5\%$ and $\sigma < 5\%$. Datasets with a larger mean ($5\% \leq \mu < 20\%$) and a relatively small coefficient of variation ($\sigma/\mu \leq 1$) are classified as having regular weight change. The weights of these datasets steadily change over time without any extreme changes. Remaining datasets are classified as having spiky weight change. In those datasets weights change drastically ($\mu > 20\%$), or there is large variation ($\sigma/\mu > 1$) along with substantial changes ($\mu > 5\%$ or $\sigma > 5\%$).

**Insertions and deletions** We distinguish between 3 subclasses: low insertions and deletions (LID), regular insertions and deletions (RID), and spiky insertions and deletions (SID). We measure the impact of insertions and deletions between two time steps $t$ and $t+1$ as the cardinality of the symmetric difference between the two sets of rectangles with non-zero weights at $t$ and $t+1$ divided by the number of rectangles with non-zero weights at $t$. We again classify the datasets based on the distribution ($\mu$ and $\sigma$) of impact values over all time steps. Same as for the weight change, LID is defined by $\mu < 5\%$ and $\sigma < 5\%$, RID is defined by $\mu < 20\%$ and $\sigma/\mu \leq 1$, and the remaining datasets are in SID.

The full classification results in $3 \times 2 \times 3 \times 3 = 54$ data classes. In Section 2.6 we evaluate how the performance of treemapping algorithms depends on the classes, that is, if the classification is sensible.

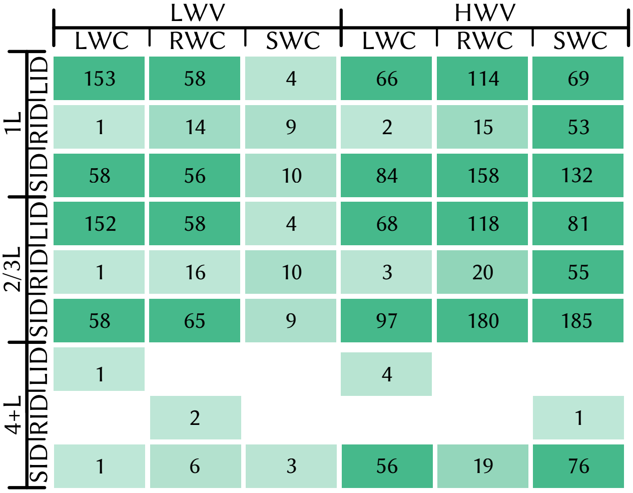
### 2.5.3 Datasets

We collected a total of 2405 time-varying hierarchical datasets from a variety of sources, detailed below. We found at least one dataset for 46 (out of 54) instances of our proposed data classes. See Figure 2.3 for the distribution of datasets over classes: clearly not all classes arise with equal frequency in our data sources.

**World Bank [158]:** (2142 datasets) World development indicators such as agriculture, rural and urban development, education, trade, and health. Hierarchy either according to the World Bank regional classification grouping countries into subregions and continents, or no hierarchy present.

**GitHub [46]:** (150 datasets) Hierarchies of folders, files, and classes, weighted by the number of code lines extracted from all revisions of several GitHub repositories using Scitools [109].

**Movies:** (107 datasets) Movies from MovieLens [57] and *TMDB* [136]. We constructed a time-varying hierarchy using the group-rows-by-attribute-value partitioning method [135, 150]. The hierarchy groups movies based on their



| | | LWV | | | HWV | | |
|---|---|---|---|---|---|---|---|
| | | LWC | RWC | SWC | LWC | RWC | SWC |
| 1L | LID | 153 | 58 | 4 | 66 | 114 | 69 |
| | RID | 1 | 14 | 9 | 2 | 15 | 53 |
| | SID | 58 | 56 | 10 | 84 | 158 | 132 |
| 2/3L | LID | 152 | 58 | 4 | 68 | 118 | 81 |
| | RID | 1 | 16 | 10 | 3 | 20 | 55 |
| | SID | 58 | 65 | 9 | 97 | 180 | 185 |
| 4+L | LID | 1 | | | 4 | | |
| | RID | | 2 | | | | 1 |
| | SID | 1 | 6 | 3 | 56 | 19 | 76 |

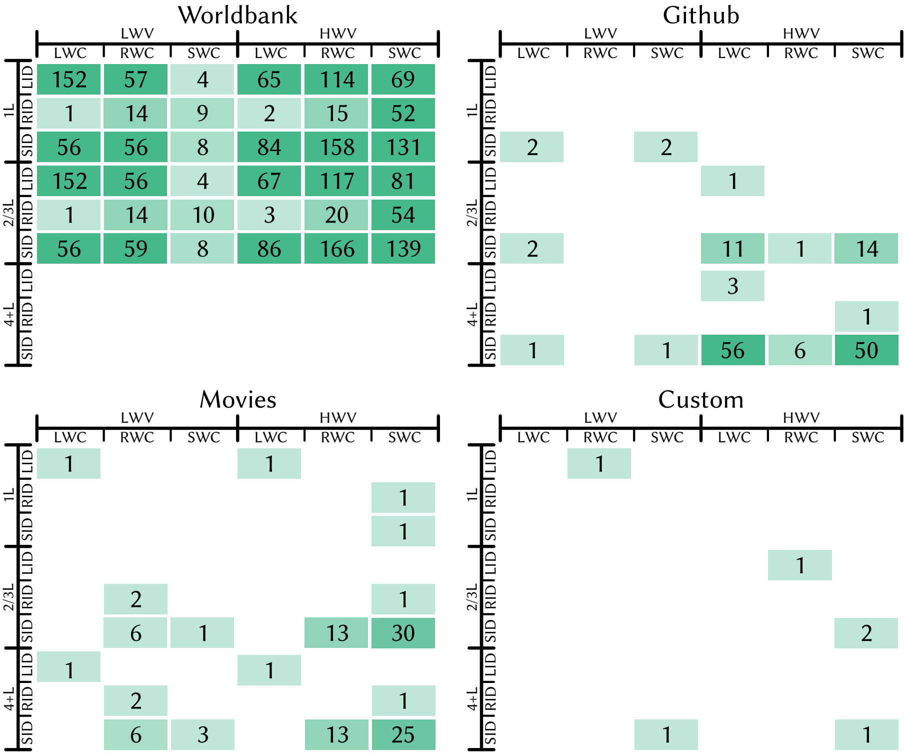
**Figure 2.3** Distribution of datasets over classes.

genres, actors, release date, and keywords. Each leaf is a movie whose weight corresponds to its rating over a given period of time.

**Custom:** (6 datasets) Several individual datasets were added: *Dutch Names* [84] contains the frequency of popular baby names in the Netherlands per year; *UN Comtrade Coffee* [144] contains the amount of coffee each country imported per year; *ATP* contains personal information, historical rankings, and match results from 1968 to 2018 for ATP tennis players [5]; and *Earthquakes* contains the time, location, depth, and intensity of seismic phenomena provided by the USGS Earthquake Hazards Program [145].

Importantly, note that the selection of dataset sources is *orthogonal* to the description of the feature space $\mathcal{F}$. The former covers the *origin* of data (which may cover application-specific aspects not captured by our feature space); the latter covers application-independent data aspects as captured by the data classes of $\mathcal{F}$.

For each data source we show the distribution of the datasets over the different data classes in Figures 2.4. In total there are 2142 datasets from the World Bank, 150 from GitHub, 107 from Movies, and 6 from Custom. The large collection of World Bank datasets contains at least one dataset for each data class with at most 3 levels of hierarchy (to which it is inherently limited), and a large enough sample for most of them for the purpose of our experiments. The GitHub and Movies datasets fill in the



Worldbank

| | | LWV | | | HWV | | |
|---|---|---|---|---|---|---|---|
| | | LWC | RWC | SWC | LWC | RWC | SWC |
| 1L | LID | 152 | 57 | 4 | 65 | 114 | 69 |
| | RID | 1 | 14 | 9 | 2 | 15 | 52 |
| | SID | 56 | 56 | 8 | 84 | 158 | 131 |
| 2/3L | LID | 152 | 56 | 4 | 67 | 117 | 81 |
| | RID | 1 | 14 | 10 | 3 | 20 | 54 |
| | SID | 56 | 59 | 8 | 86 | 166 | 139 |
| 4+L | LID | | | | | | |
| | RID | | | | | | |
| | SID | | | | | | |

Github

| | | LWV | | | HWV | | |
|---|---|---|---|---|---|---|---|
| | | LWC | RWC | SWC | LWC | RWC | SWC |
| 1L | LID | | | | | | |
| | RID | | | | | | |
| | SID | 2 | | 2 | | | |
| 2/3L | LID | | | | 1 | | |
| | RID | | | | | | |
| | SID | 2 | | | 11 | 1 | 14 |
| 4+L | LID | | | | 3 | | |
| | RID | | | | | | 1 |
| | SID | 1 | | 1 | 56 | 6 | 50 |

Movies

| | | LWV | | | HWV | | |
|---|---|---|---|---|---|---|---|
| | | LWC | RWC | SWC | LWC | RWC | SWC |
| 1L | LID | 1 | | | 1 | | |
| | RID | | | | | | 1 |
| | SID | | | | | | 1 |
| 2/3L | LID | | | | | | |
| | RID | | 2 | | | | 1 |
| | SID | | 6 | 1 | | 13 | 30 |
| 4+L | LID | 1 | | | 1 | | |
| | RID | | 2 | | | | 1 |
| | SID | | 6 | 3 | | 13 | 25 |

Custom

| | | LWV | | | HWV | | |
|---|---|---|---|---|---|---|---|
| | | LWC | RWC | SWC | LWC | RWC | SWC |
| 1L | LID | | 1 | | | | |
| | RID | | | | | | |
| | SID | | | | | | |
| 2/3L | LID | | | | | 1 | |
| | RID | | | | | | |
| | SID | | | | | | 2 |
| 4+L | LID | | | | | | |
| | RID | | | | | | |
| | SID | | | 1 | | | 1 |

**Figure 2.4** Distribution of the datasets. From left to right and top to bottom: Worldbank, Github, Movies, Custom.

remaining data classes (with 4+ levels of hierarchy) for which we have data.

## 2.6 Experimental results

We ran all 14 rectangular treemapping algorithms (Section 2.1) on all time steps of all 2405 datasets, generated the baselines for all these instances (Section 2.4), and recorded all resulting layouts. Per dataset we aggregate our results for all metrics and algorithms by first taking the mean over all rectangles in a single time step, and then taking the mean again over all time steps. This is necessary since the number of rectangles may differ per time step.

We focus on three specific questions: we first explore the *validity* of our data classification (Section 2.6.1) and then we study the *performance* of all algorithms with respect to visual quality and stability across varying data features (Section 2.6.2). Finally, we compare the performance of all algorithms on each data *class* separately (Section 2.6.3). We believe that the resulting visual summary will help researchers and practitioners choose a suitable treemapping algorithm for their data.

### 2.6.1 Data classification analysis

We evaluate if the relative performance of treemapping algorithms is more consistent within a data class than for an arbitrary collection of datasets. To perform this analysis we need to establish how we can capture the consistency of relative performance for a collection of datasets and how we can compare this consistency between multiple collections. We restrict our analysis to data classes that contain at least 50 datasets, for otherwise the observed consistency is not sufficiently reliable. For each such data class we randomly sample 50 datasets to use in this analysis. We also randomly sample 50 datasets among all 2405 datasets (all classes) as a baseline for comparison. Note that all collections must have the same number of datasets in the analysis to ensure that the comparisons are fair.

Now consider a single collection of datasets. To measure the consistency of relative performance among different datasets in this collection we cannot directly use the computed metrics for visual quality and stability, as these values may differ greatly between datasets. Alternatively, we could rank the algorithms per dataset, but then algorithms with very similar performance may imply a greater variance in relative performance than is the case. Instead, we define the *relative performance* (separately for visual quality and stability) per dataset as follows. We compute both the best value (maximum for visual quality, minimum for stability) and the median value over all algorithms over this dataset. The *relative performance score* for each algorithm on this dataset is then computed by linearly interpolating between these two values where the best algorithm receives score 0, and the median algorithm receives score 0.5. The relative performance score is capped at 1, to avoid outliers. The resulting scores are comparable between different datasets.

We next analyze the consistency of relative performance within collections of datasets in two ways. First, we use a quantitative approach: for each algorithm we compute the variance of the relative performance scores over all datasets in a collection and sum up the variances over all algorithms. This results in a *consistency score* $c$ for a collection of datasets. Figure 2.5 displays the consistency scores (for visual quality

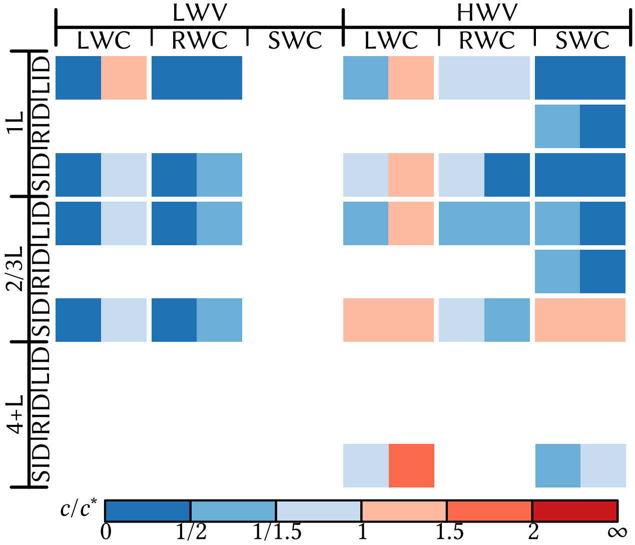


**Figure 2.5** Ratio of the consistency score (Left: Visual quality, Right: Stability) between the data class and the baseline for each data class with at least 50 datasets.

and stability) of all data classes (with at least 50 datasets) compared to the consistency scores $c^*$ of the baseline collection (created by random sampling). A cell is colored blue (more consistent) if $c < c^*$; a cell is colored red (less consistent) if $c > c^*$.

Nearly all data classes for visual quality and most data classes for stability are more consistent than the baseline. This indicates that our features are splitting the datasets into valid data classes where the relative performance of an algorithm is easier to predict than in the baseline. However, the stability column for high weight variance and low weight change is less consistent than the baseline. As discussed in Section 2.1, the stability of unordered treemaps becomes worse compared to ordered treemaps when the weight variance is low or the weight change is high due to reordering of the input weights. As a result the difference with respect to stability between ordered and unordered treemaps is less pronounced for these data classes; the relative performance is hence influenced more by accidental details of individual datasets and less by structural differences between the algorithms. Additionally, there are two data classes where the visual quality is less consistent than the baseline. It is not clear to us at this point what the cause of these inconsistencies is; one possibility are hidden correlations in the data classes.

Second, we use a more qualitative approach to assess the consistency of relative performance. For each data class we create a matrix plot that shows the relative performance scores of all algorithms for all datasets in the collection (see Figures 2.6 and 2.7). Each column in the matrix plot represents a dataset, and each row represents an algorithm. The color of every "cell" in the matrix plot indicates the relative performance score of an algorithm on a dataset where lighter colors indicate better (lower) relative performance scores. Relative performance scores that were capped at 1 are indicated with purple. To better enable the visual assessment of consistency among the different datasets in a collection, we order the datasets (columns) so that those with similar scores are next to each other as much as possible. Also, we order the algorithms (rows) so that the algorithms with better average score are lower in the matrix plot. In particular, the order of algorithms in the matrix plots for different data classes can be different. Figure 2.6 shows the matrix plots for visual quality with the corresponding matrix plot for the baseline collection at the left-top. Figure 2.7 shows the matrix plots for stability.

First consider the matrix plot for visual quality (Figure 2.6). For the low weight variance subclass we indeed see that the matrix plots are much smoother than the baseline which confirms the results in Figure 2.5. We also observe an increasing number of irregularities when going from 1 level treemaps to 2/3 levels or 4+ levels, since more levels impose more restrictions on the layout and hence all algorithms perform more similarly.

Consider now Figure 2.7. First of all, we notice that there is a set of four algorithms at the bottom of every matrix plot. These are the state-aware algorithms (LM0, LM4, and GIT) and SND. For nearly all datasets, regardless of the specific data class, these four algorithms are much more stable than any of the others. There is also a large difference between the low weight variance and high weight variance subclasses. For low weight variance there is a set of algorithms that perform consistently much worse than the median (purple cells). These include the unordered treemaps which are particularly sensitive to changes in such data.

### 2.6.2 Performance analysis across features

The analysis in Section 2.6.1 shows that our data classification is valid. We now study how visual quality and stability depend on the *features* of the datasets. We aim to understand how sensitive a given algorithm is to variations in one or several features of the input data. For each data class we calculate the average visual quality and stability. For each subclass of a feature we then take the average over all data

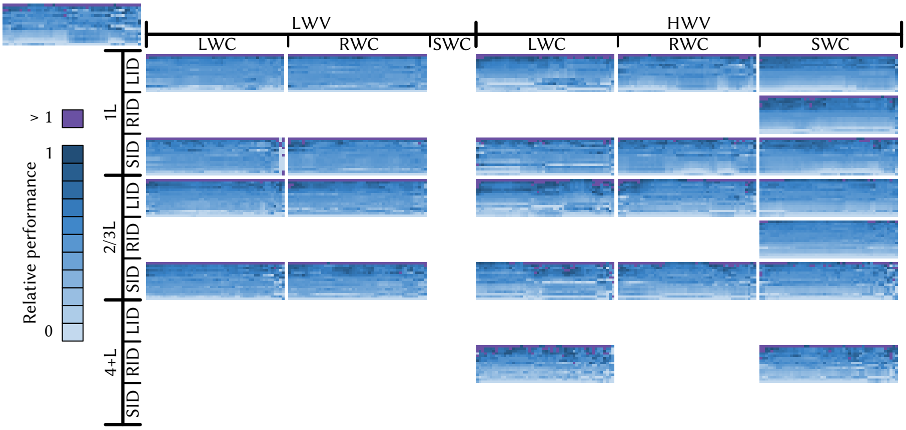


**Figure 2.6** Visual quality: matrix plots for each data class with at least 50 datasets plus baseline (left top). In each matrix plot, rows correspond to algorithms, columns to datasets. The lighter the color, the better the relative performance, capped at 1 (purple).

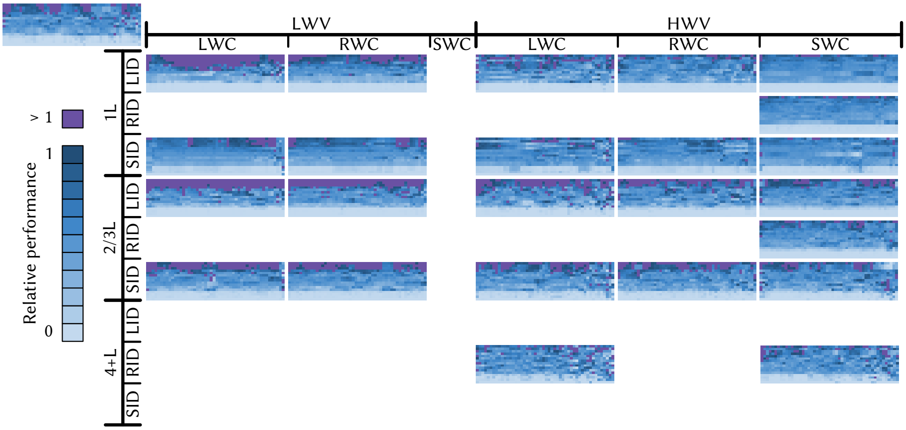


**Figure 2.7** Stability: matrix plots for each data class with at least 50 datasets plus baseline (left top). In each matrix plot, rows correspond to algorithms, columns to datasets. The lighter the color, the better the relative performance, capped at 1 (purple).

classes that belong to it. This ensures that even though we have different numbers of datasets in each data class they are all weighted equally. We show this data in Figures 2.8–2.11. Each point in a figure represents the score for one algorithm on one subclass of the feature, for example, low weight variance. A polyline is drawn that connects the points of one algorithm and we use glyphs to indicate the different subclasses of the feature. The different algorithms are indicated with categorical colors (see figure legends).

Recall that a low value for the stability metric indicates a *stable* algorithm and that the visual quality metric (aspect ratio) is bounded between 0 and 1. In particular, note that visual quality ($\rho$) of 0.5 for a rectangle indicates a 2-by-1 rectangle. A $\rho$ of 0.25 is perceptually much worse than a $\rho$ of 0.5 in terms of area perception as can be inferred from Kong *et al.* [70] coming close to their "extreme aspect ratios" of 4.5.

**Levels of hierarchy** Figure 2.8 considers the levels of hierarchy feature. In this figure we observe that all algorithms, in particular the stateless ones, are more stable as the number of levels increase. In contrast to most other algorithms, the visual quality of state-aware algorithms (LM0, LM4, GIT) as well as SND increases with the number of levels. We also see that SQR and PBS have the longest polylines, that is, they are the most sensitive to the number of levels.

**Variance of node weights** Figure 2.9 considers the weight variance feature. Increasing the weight variance decreases the visual quality for all algorithms except for APP (and SND). Additionally, we see that the unordered treemaps are indeed more sensitive to this feature in terms of stability compared to the other algorithms. These algorithms reorder the data based on the weight to determine their layout, and if the weight are close to each other this happen more often.

**Weight change** Figure 2.10 considers the weight change feature. which has three values: LWC, RWC, and SWC. The near-vertical polylines for the stateless algorithms show that visual quality seems to be largely unaffected by this feature. The stability however decreases quickly. Conversely, for the state-aware algorithms the polylines are mostly near-horizontal: the stability is largely unaffected, but the visual quality decreases. As the only state-aware algorithm that allows changes to the layout, LM4 makes an explicit trade-off between stability and visual quality (see the slightly sloping line).

**Insertions and deletions** Finally, Figure 2.11 considers the insertions and deletions feature. The plot shows a similar variation of visual quality and stability as seen

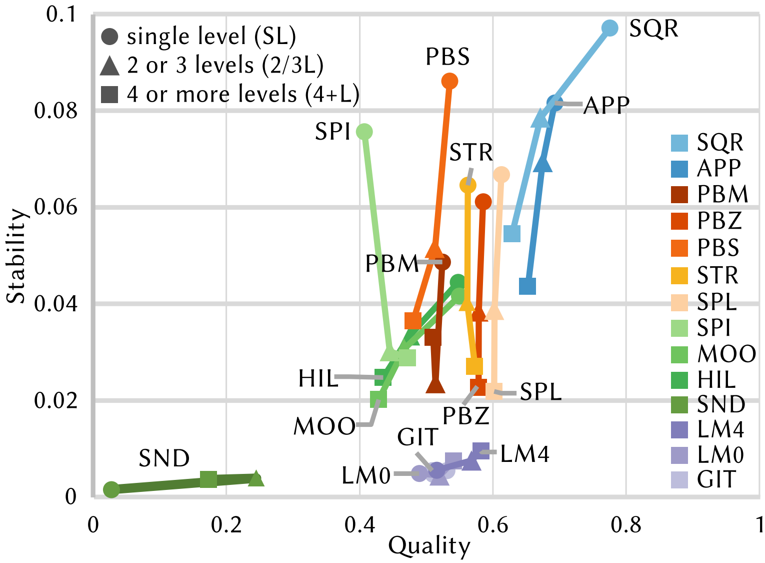


**Figure 2.8** Visual quality *vs* stability as function of the levels of hierarchy feature.

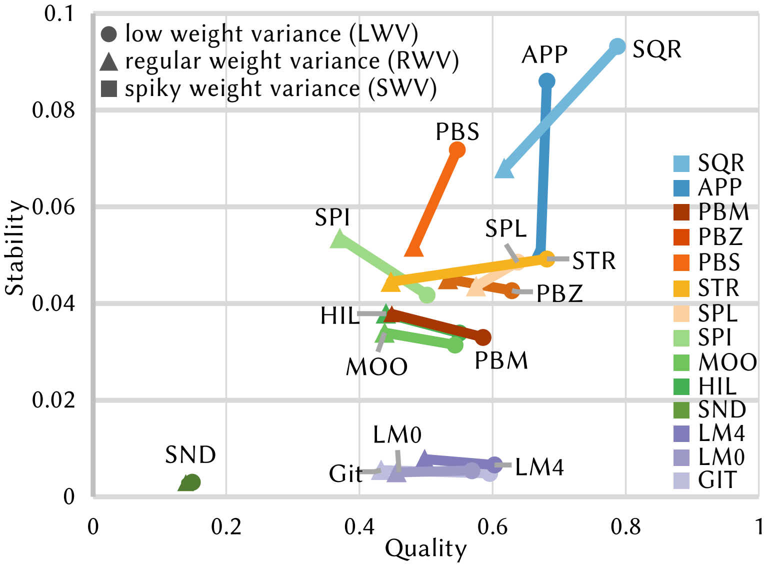


**Figure 2.9** Visual quality *vs* stability as function of the variance of node weights feature.

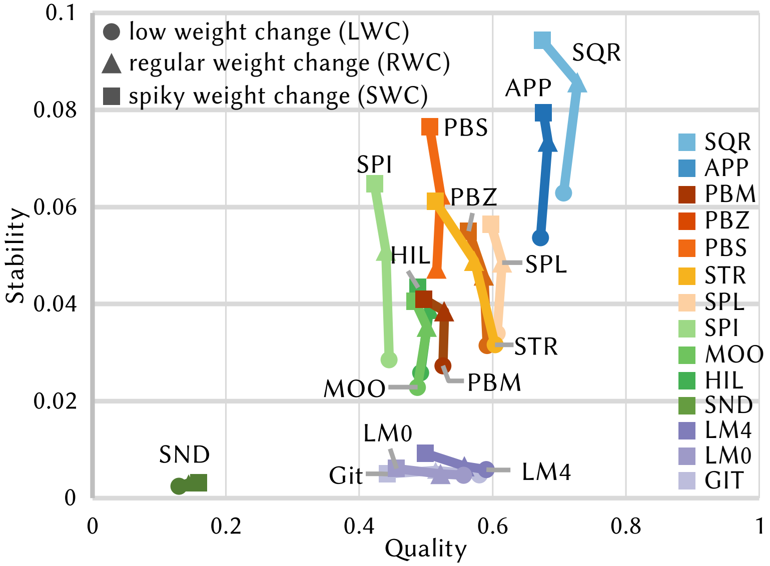


**Figure 2.10** Visual quality *vs* stability as function of the weight change feature.

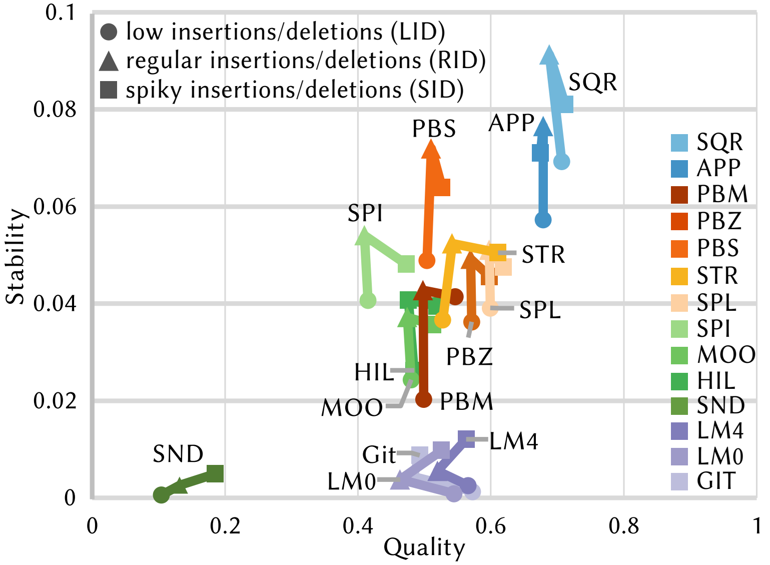


**Figure 2.11** Visual quality *vs* stability as function of the insertions and deletions feature.

for the weight change feature (Figure 2.10). Yet, the polylines for the stateless algorithms now show a ‘kink’ at the midpoint (RID, regular insertions/deletions). Hence, these algorithms are most unstable for regular insertions/deletions, and stabler for linear and spiky insertions/deletions. Interestingly, the state-aware methods (LM0, LM4, and GIT) show a similar kink but oriented differently. These methods thus achieve poorest visual quality for regular insertions/deletions and highest quality on the other two values of this feature.

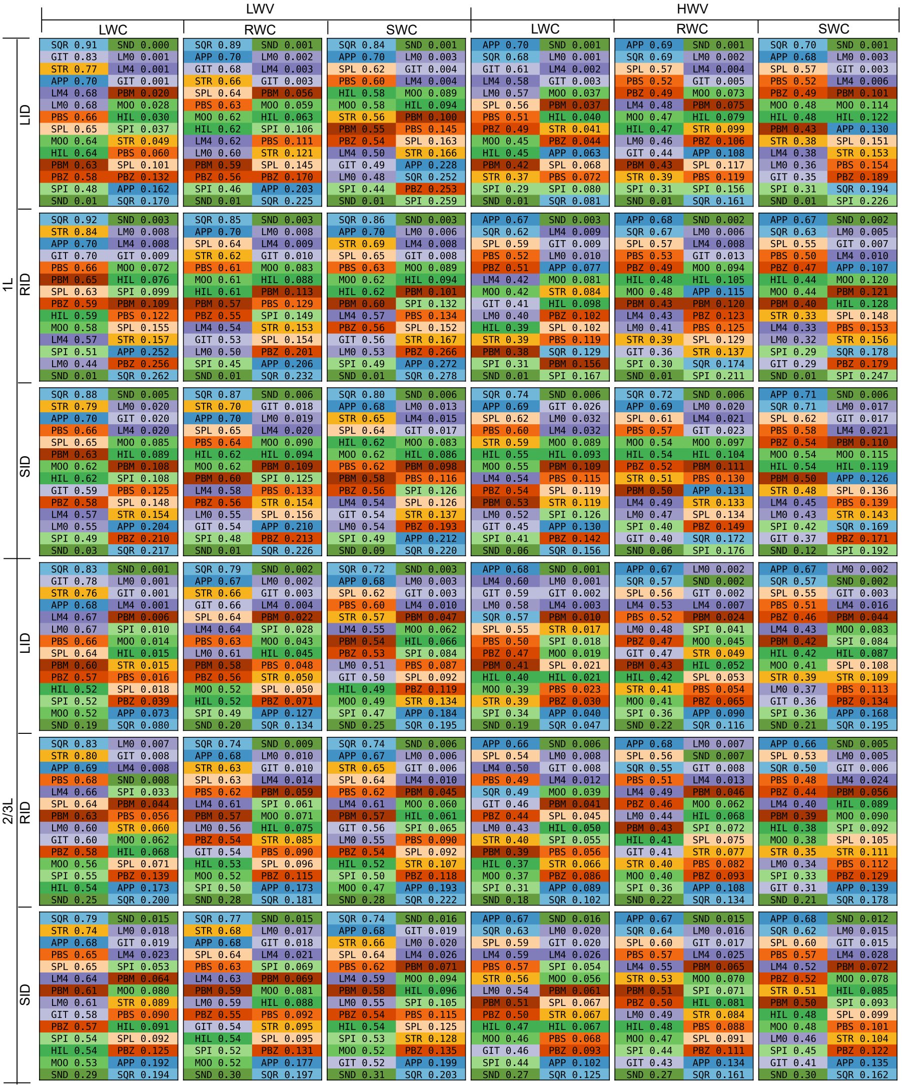
### 2.6.3 Comparison of data classes

We compare the relative performance of all algorithms separately on all data classes. Figures 2.12 and 2.13 supports this comparison as follows: it is structured as a matrix of tables, one per data class. Each table shows the average visual quality (left column) and average stability (right column) of all algorithms for all datasets in the respective data class. The two columns are sorted separately to show the best-ranking algorithms at the top. Cells show the algorithm names and scores, and are categorically color-coded on the algorithm name following the same color scheme as in Section 2.6.2. Empty cells indicate data classes for which we did not find datasets. Figures 2.12 and 2.13 can answer the following practical questions:

**Which method is best for my data?** Given a family of datasets with known characteristics (feature values) we search for the corresponding cell and pick the top algorithm(s) in visual quality, stability, or a combination of both depending on the application requirements. When doing this we should examine the actual values, since several algorithms score quite close to each other.

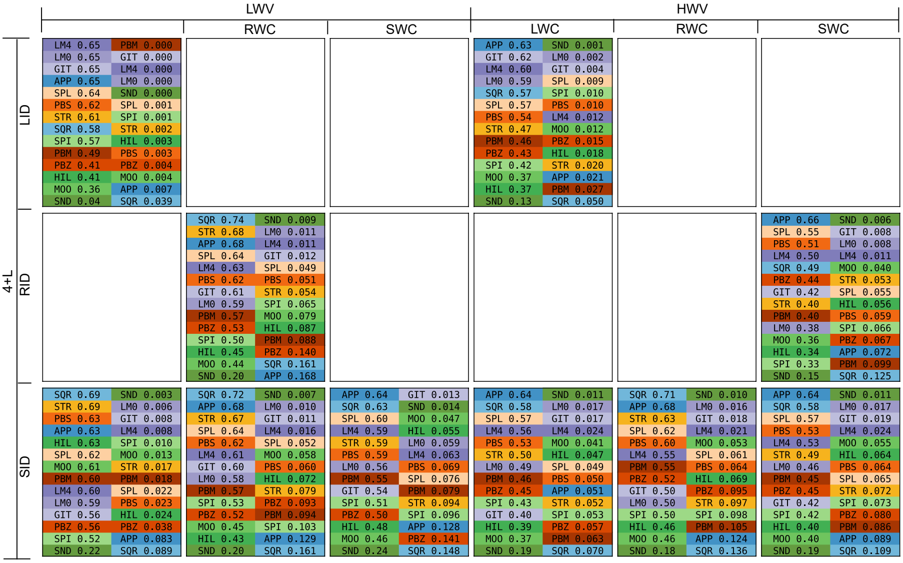
**How is a given algorithm performing in general?** We scan the table following the color of the respective algorithm, and detect its rank with respect to visual quality and/or stability over all data classes. In this way we can find patterns and outliers in the data for this algorithm: for example, LM0 and LM4 are always near the top in stability, and GIT’s visual quality fluctuates widely depending on the data class.

**Which algorithms perform similarly?** We locate groups of neighboring rows with the same color pattern in all tables. These groups indicate algorithms which score similarly regardless of data class.



| | | LWV | | | | | | HWV | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LWC | | RWC | | SWC | | LWC | | RWC | | SWC | |
| 1L | LID | SQR 0.91 | SND 0.000 | SQR 0.89 | SND 0.001 | SQR 0.84 | SND 0.001 | APP 0.70 | SND 0.001 | APP 0.69 | SND 0.001 | SQR 0.70 | SND 0.001 |
| | | GIT 0.83 | LM0 0.001 | APP 0.70 | LM0 0.002 | APP 0.70 | LM0 0.003 | SQR 0.68 | LM0 0.001 | SQR 0.69 | LM0 0.002 | APP 0.68 | LM0 0.003 |
| | | STR 0.77 | LM4 0.001 | GIT 0.68 | LM4 0.003 | SPL 0.62 | GIT 0.004 | GIT 0.61 | LM4 0.002 | SPL 0.57 | LM4 0.004 | SPL 0.57 | GIT 0.003 |
| | | APP 0.70 | GIT 0.001 | STR 0.66 | GIT 0.003 | PBS 0.60 | LM4 0.004 | LM4 0.58 | GIT 0.003 | PBS 0.52 | GIT 0.005 | PBS 0.52 | LM4 0.006 |
| | | LM4 0.68 | PBM 0.020 | SPL 0.64 | PBM 0.056 | HIL 0.58 | MOO 0.089 | LM0 0.57 | MOO 0.037 | PBZ 0.49 | MOO 0.073 | PBZ 0.49 | PBM 0.101 |
| | | LM0 0.68 | MOO 0.028 | PBS 0.63 | MOO 0.059 | MOO 0.58 | HIL 0.094 | SPL 0.56 | PBM 0.037 | LM4 0.48 | PBM 0.075 | MOO 0.48 | MOO 0.114 |
| | | PBS 0.66 | HIL 0.030 | MOO 0.62 | HIL 0.063 | STR 0.56 | PBM 0.100 | PBS 0.51 | HIL 0.040 | MOO 0.47 | HIL 0.079 | HIL 0.48 | HIL 0.122 |
| | | SPL 0.65 | SPI 0.037 | HIL 0.62 | SPI 0.106 | PBM 0.55 | PBS 0.145 | PBZ 0.49 | STR 0.041 | HIL 0.47 | STR 0.099 | PBM 0.43 | APP 0.130 |
| | | MOO 0.64 | STR 0.049 | LM4 0.62 | PBS 0.111 | PBZ 0.54 | SPL 0.163 | MOO 0.45 | PBZ 0.044 | LM0 0.46 | PBZ 0.106 | STR 0.38 | SPL 0.151 |
| | | HIL 0.64 | PBS 0.060 | LM0 0.60 | STR 0.121 | LM4 0.50 | STR 0.166 | HIL 0.45 | APP 0.063 | GIT 0.44 | APP 0.108 | LM4 0.38 | STR 0.153 |
| | | PBM 0.63 | SPL 0.101 | PBM 0.59 | SPL 0.145 | GIT 0.49 | APP 0.228 | PBM 0.42 | SPL 0.068 | PBM 0.43 | SPL 0.117 | LM0 0.36 | PBS 0.154 |
| | | PBZ 0.58 | PBZ 0.132 | PBZ 0.56 | PBZ 0.170 | LM0 0.48 | SQR 0.252 | STR 0.37 | PBS 0.072 | STR 0.39 | PBS 0.119 | GIT 0.35 | PBZ 0.189 |
| | | SPI 0.48 | APP 0.162 | SPI 0.46 | APP 0.203 | SPI 0.44 | PBZ 0.253 | SPI 0.29 | SPI 0.080 | SPI 0.31 | SPI 0.156 | SPI 0.31 | SQR 0.194 |
| | | SND 0.01 | SQR 0.170 | SND 0.01 | SQR 0.225 | SND 0.01 | SPI 0.259 | SND 0.01 | SQR 0.081 | SND 0.01 | SQR 0.161 | SND 0.01 | SPI 0.226 |
| | RID | SQR 0.92 | SND 0.003 | SQR 0.85 | SND 0.003 | SQR 0.86 | SND 0.003 | APP 0.67 | SND 0.003 | APP 0.68 | SND 0.002 | APP 0.67 | SND 0.002 |
| | | STR 0.84 | LM0 0.008 | APP 0.70 | LM0 0.008 | APP 0.70 | LM0 0.006 | SQR 0.62 | LM4 0.009 | SQR 0.67 | LM0 0.006 | SQR 0.63 | LM0 0.005 |
| | | APP 0.70 | LM4 0.008 | SPL 0.64 | LM4 0.009 | STR 0.69 | LM4 0.008 | SPL 0.59 | GIT 0.009 | SPL 0.57 | LM4 0.008 | SPL 0.55 | GIT 0.007 |
| | | GIT 0.70 | GIT 0.009 | STR 0.62 | GIT 0.010 | SPL 0.65 | GIT 0.008 | PBS 0.52 | LM0 0.010 | PBS 0.53 | GIT 0.013 | PBS 0.50 | LM4 0.010 |
| | | PBS 0.66 | MOO 0.072 | PBS 0.61 | MOO 0.083 | PBS 0.63 | MOO 0.089 | PBZ 0.51 | APP 0.077 | PBZ 0.49 | MOO 0.094 | PBZ 0.47 | APP 0.107 |
| | | PBM 0.65 | HIL 0.076 | MOO 0.61 | HIL 0.088 | MOO 0.62 | HIL 0.094 | LM4 0.42 | MOO 0.081 | HIL 0.48 | HIL 0.105 | HIL 0.44 | MOO 0.120 |
| | | SPL 0.63 | SPI 0.099 | HIL 0.61 | PBM 0.113 | HIL 0.62 | PBM 0.101 | MOO 0.42 | STR 0.084 | MOO 0.48 | APP 0.115 | MOO 0.44 | PBM 0.121 |
| | | PBZ 0.59 | PBM 0.109 | PBM 0.57 | PBS 0.129 | PBM 0.60 | SPI 0.132 | GIT 0.41 | HIL 0.098 | PBM 0.43 | PBM 0.120 | PBM 0.40 | HIL 0.128 |
| | | HIL 0.59 | PBS 0.122 | PBZ 0.55 | SPI 0.149 | LM4 0.57 | PBS 0.134 | LM0 0.40 | PBZ 0.102 | LM4 0.43 | PBZ 0.123 | STR 0.33 | SPL 0.148 |
| | | MOO 0.58 | SPL 0.155 | LM4 0.54 | STR 0.153 | PBZ 0.56 | SPL 0.152 | HIL 0.39 | SPL 0.102 | LM0 0.41 | PBS 0.125 | LM4 0.33 | PBS 0.153 |
| | | LM4 0.57 | STR 0.157 | GIT 0.53 | SPL 0.154 | GIT 0.56 | STR 0.167 | STR 0.39 | PBS 0.119 | STR 0.39 | SPL 0.129 | LM0 0.32 | STR 0.156 |
| | | SPI 0.51 | APP 0.252 | LM0 0.50 | PBZ 0.201 | LM0 0.53 | PBZ 0.266 | PBM 0.38 | SQR 0.129 | GIT 0.36 | STR 0.137 | SPI 0.29 | SQR 0.178 |
| | | LM0 0.44 | PBZ 0.256 | SPI 0.45 | APP 0.206 | SPI 0.49 | APP 0.272 | SPI 0.31 | PBM 0.156 | SPI 0.30 | SQR 0.174 | GIT 0.29 | PBZ 0.179 |
| | | SND 0.01 | SQR 0.262 | SND 0.01 | SQR 0.232 | SND 0.01 | SQR 0.278 | SND 0.01 | SPI 0.167 | SND 0.01 | SPI 0.211 | SND 0.01 | SPI 0.247 |
| | SID | SQR 0.88 | SND 0.005 | SQR 0.87 | SND 0.006 | SQR 0.80 | SND 0.006 | SQR 0.74 | SND 0.006 | SQR 0.72 | SND 0.006 | APP 0.71 | SND 0.006 |
| | | STR 0.79 | LM0 0.020 | STR 0.70 | GIT 0.018 | APP 0.68 | LM0 0.013 | APP 0.69 | GIT 0.026 | APP 0.69 | LM0 0.020 | SQR 0.71 | LM0 0.017 |
| | | APP 0.70 | GIT 0.020 | APP 0.70 | LM0 0.019 | STR 0.65 | LM4 0.015 | SPL 0.62 | LM0 0.032 | SPL 0.61 | LM4 0.021 | SPL 0.62 | GIT 0.017 |
| | | PBS 0.66 | LM4 0.020 | SPL 0.65 | LM4 0.020 | SPL 0.64 | GIT 0.017 | PBS 0.60 | LM4 0.032 | PBS 0.57 | GIT 0.023 | PBS 0.58 | LM4 0.021 |
| | | SPL 0.65 | MOO 0.085 | PBS 0.64 | MOO 0.090 | HIL 0.62 | MOO 0.083 | STR 0.59 | MOO 0.089 | MOO 0.54 | MOO 0.097 | PBZ 0.54 | PBM 0.110 |
| | | PBM 0.63 | HIL 0.089 | HIL 0.62 | HIL 0.094 | MOO 0.62 | HIL 0.086 | HIL 0.55 | HIL 0.093 | HIL 0.54 | HIL 0.104 | MOO 0.54 | MOO 0.115 |
| | | MOO 0.62 | PBM 0.108 | MOO 0.62 | PBM 0.109 | PBS 0.62 | PBM 0.098 | MOO 0.55 | PBM 0.109 | PBZ 0.52 | PBM 0.111 | HIL 0.54 | HIL 0.119 |
| | | HIL 0.62 | SPI 0.108 | PBM 0.60 | SPI 0.125 | PBM 0.58 | PBS 0.116 | LM4 0.54 | PBS 0.115 | STR 0.51 | PBS 0.130 | PBM 0.50 | APP 0.126 |
| | | GIT 0.59 | PBS 0.125 | LM4 0.58 | PBS 0.133 | PBZ 0.56 | SPI 0.126 | PBZ 0.54 | SPL 0.119 | PBM 0.50 | APP 0.131 | STR 0.48 | SPL 0.136 |
| | | PBZ 0.58 | SPL 0.148 | PBZ 0.56 | STR 0.154 | LM4 0.54 | SPL 0.126 | PBM 0.53 | STR 0.119 | LM4 0.49 | STR 0.133 | LM4 0.45 | PBS 0.139 |
| | | LM4 0.57 | STR 0.154 | LM0 0.55 | SPL 0.156 | GIT 0.54 | STR 0.137 | LM0 0.52 | SPI 0.126 | LM0 0.47 | SPL 0.134 | LM0 0.43 | STR 0.143 |
| | | LM0 0.55 | APP 0.204 | GIT 0.54 | APP 0.210 | LM0 0.54 | PBZ 0.193 | GIT 0.45 | APP 0.130 | SPI 0.40 | PBZ 0.149 | SPI 0.42 | SQR 0.169 |
| | | SPI 0.49 | PBZ 0.210 | SPI 0.48 | PBZ 0.213 | SPI 0.49 | APP 0.212 | SPI 0.41 | PBZ 0.142 | GIT 0.40 | SQR 0.172 | GIT 0.37 | PBZ 0.171 |
| | | SND 0.03 | SQR 0.217 | SND 0.01 | SQR 0.226 | SND 0.09 | SQR 0.220 | SND 0.06 | SQR 0.156 | SND 0.06 | SPI 0.176 | SND 0.12 | SPI 0.192 |
| 2/3L | LID | SQR 0.83 | SND 0.001 | SQR 0.79 | SND 0.002 | SQR 0.72 | SND 0.003 | APP 0.68 | SND 0.001 | APP 0.67 | LM0 0.002 | APP 0.67 | LM0 0.002 |
| | | GIT 0.78 | LM0 0.001 | APP 0.67 | LM0 0.002 | APP 0.68 | LM0 0.003 | LM4 0.60 | LM0 0.001 | SQR 0.57 | SND 0.002 | SQR 0.57 | SND 0.002 |
| | | STR 0.76 | GIT 0.001 | STR 0.66 | GIT 0.003 | SPL 0.62 | GIT 0.003 | GIT 0.59 | GIT 0.002 | SPL 0.56 | GIT 0.002 | SPL 0.55 | GIT 0.003 |
| | | APP 0.68 | LM4 0.001 | GIT 0.66 | LM4 0.004 | PBS 0.60 | LM4 0.010 | LM0 0.58 | LM4 0.003 | LM4 0.53 | LM4 0.007 | PBS 0.51 | LM4 0.016 |
| | | LM4 0.67 | PBM 0.006 | SPL 0.64 | PBM 0.022 | STR 0.57 | PBM 0.047 | SQR 0.57 | PBM 0.010 | PBS 0.52 | PBM 0.024 | PBZ 0.46 | PBM 0.044 |
| | | LM0 0.67 | SPI 0.010 | LM4 0.64 | SPI 0.028 | LM4 0.55 | MOO 0.062 | SPL 0.55 | STR 0.017 | LM0 0.48 | SPI 0.041 | LM4 0.43 | MOO 0.083 |
| | | PBS 0.66 | MOO 0.014 | PBS 0.63 | MOO 0.043 | PBM 0.54 | HIL 0.066 | PBS 0.50 | SPI 0.018 | PBZ 0.47 | MOO 0.045 | PBM 0.42 | SPI 0.084 |
| | | SPL 0.64 | HIL 0.015 | LM0 0.61 | HIL 0.045 | PBZ 0.53 | SPI 0.084 | PBZ 0.47 | MOO 0.019 | GIT 0.47 | STR 0.049 | HIL 0.42 | HIL 0.087 |
| | | PBM 0.60 | STR 0.015 | PBM 0.58 | PBS 0.048 | LM0 0.51 | PBS 0.087 | PBM 0.41 | SPL 0.021 | PBM 0.43 | HIL 0.052 | MOO 0.41 | SPL 0.108 |
| | | PBZ 0.57 | PBS 0.016 | PBZ 0.56 | STR 0.050 | GIT 0.50 | SPL 0.092 | HIL 0.40 | HIL 0.021 | HIL 0.42 | SPL 0.053 | STR 0.39 | STR 0.109 |
| | | HIL 0.52 | SPL 0.018 | MOO 0.52 | SPL 0.050 | HIL 0.49 | PBZ 0.119 | MOO 0.39 | PBS 0.023 | STR 0.41 | PBS 0.054 | LM0 0.37 | PBS 0.113 |
| | | SPI 0.52 | PBZ 0.039 | HIL 0.52 | PBZ 0.071 | MOO 0.49 | STR 0.134 | STR 0.39 | PBZ 0.030 | MOO 0.41 | PBZ 0.065 | GIT 0.36 | PBZ 0.134 |
| | | MOO 0.52 | APP 0.073 | SPI 0.49 | APP 0.127 | SPI 0.47 | APP 0.184 | SPI 0.34 | APP 0.040 | SPI 0.36 | APP 0.090 | SPI 0.36 | APP 0.168 |
| | | SND 0.19 | SQR 0.080 | SND 0.20 | SQR 0.134 | SND 0.25 | SQR 0.195 | SND 0.19 | SQR 0.047 | SND 0.22 | SQR 0.116 | SND 0.21 | SQR 0.195 |
| | RID | SQR 0.83 | LM0 0.007 | SQR 0.74 | SND 0.009 | SQR 0.74 | SND 0.006 | APP 0.66 | SND 0.006 | APP 0.68 | LM0 0.007 | APP 0.66 | SND 0.005 |
| | | STR 0.80 | GIT 0.008 | APP 0.68 | LM0 0.010 | APP 0.67 | LM0 0.006 | SPL 0.54 | LM0 0.008 | SPL 0.56 | SND 0.007 | SPL 0.53 | LM0 0.005 |
| | | APP 0.69 | LM4 0.008 | STR 0.63 | GIT 0.010 | STR 0.65 | GIT 0.006 | LM4 0.50 | GIT 0.008 | SQR 0.55 | GIT 0.008 | SQR 0.50 | GIT 0.006 |
| | | PBS 0.68 | SND 0.008 | SPL 0.63 | LM4 0.014 | SPL 0.64 | LM4 0.010 | PBS 0.49 | LM4 0.012 | PBS 0.51 | LM4 0.013 | PBS 0.48 | LM4 0.024 |
| | | LM4 0.66 | SPI 0.033 | PBS 0.62 | PBM 0.059 | PBS 0.62 | PBM 0.045 | SQR 0.49 | MOO 0.039 | LM4 0.49 | PBM 0.046 | PBZ 0.44 | PBM 0.056 |
| | | SPL 0.64 | PBM 0.044 | LM4 0.61 | SPI 0.061 | LM4 0.61 | MOO 0.060 | GIT 0.46 | PBM 0.041 | PBZ 0.46 | MOO 0.062 | LM4 0.40 | HIL 0.089 |
| | | PBM 0.63 | PBS 0.056 | PBM 0.57 | MOO 0.071 | PBM 0.57 | HIL 0.061 | PBZ 0.44 | SPL 0.045 | LM0 0.44 | HIL 0.068 | PBM 0.39 | MOO 0.090 |
| | | LM0 0.60 | STR 0.060 | LM0 0.56 | HIL 0.075 | GIT 0.56 | SPI 0.065 | LM0 0.43 | HIL 0.050 | PBM 0.43 | SPI 0.072 | HIL 0.38 | SPI 0.092 |
| | | GIT 0.60 | MOO 0.062 | PBZ 0.54 | STR 0.085 | LM0 0.55 | PBS 0.090 | STR 0.40 | SPI 0.055 | HIL 0.41 | SPL 0.075 | MOO 0.38 | SPL 0.105 |
| | | PBZ 0.58 | HIL 0.068 | GIT 0.54 | PBS 0.090 | PBZ 0.54 | SPL 0.092 | PBM 0.39 | PBS 0.056 | GIT 0.41 | STR 0.077 | STR 0.35 | STR 0.111 |
| | | MOO 0.56 | SPL 0.071 | HIL 0.53 | SPL 0.096 | HIL 0.52 | STR 0.107 | HIL 0.37 | STR 0.066 | STR 0.40 | PBS 0.082 | LM0 0.34 | PBS 0.112 |
| | | SPI 0.55 | PBZ 0.139 | MOO 0.52 | PBZ 0.115 | SPI 0.50 | PBZ 0.118 | MOO 0.37 | PBZ 0.086 | MOO 0.40 | PBZ 0.093 | SPI 0.33 | PBZ 0.129 |
| | | HIL 0.54 | APP 0.173 | SPI 0.50 | APP 0.173 | MOO 0.47 | APP 0.193 | SPI 0.31 | APP 0.089 | SPI 0.36 | APP 0.108 | GIT 0.31 | APP 0.139 |
| | | SND 0.25 | SQR 0.200 | SND 0.28 | SQR 0.181 | SND 0.28 | SQR 0.222 | SND 0.18 | SQR 0.102 | SND 0.22 | SQR 0.134 | SND 0.21 | SQR 0.178 |
| | SID | SQR 0.79 | SND 0.015 | SQR 0.77 | SND 0.015 | SQR 0.74 | SND 0.016 | APP 0.67 | SND 0.016 | APP 0.67 | SND 0.015 | APP 0.68 | SND 0.012 |
| | | STR 0.74 | LM0 0.018 | STR 0.68 | LM0 0.017 | APP 0.68 | GIT 0.019 | SQR 0.63 | LM0 0.020 | SQR 0.64 | LM0 0.016 | SQR 0.62 | LM0 0.015 |
| | | APP 0.68 | GIT 0.019 | APP 0.68 | GIT 0.018 | STR 0.66 | LM0 0.020 | SPL 0.59 | GIT 0.020 | SPL 0.60 | GIT 0.017 | SPL 0.60 | GIT 0.015 |
| | | PBS 0.65 | LM4 0.023 | SPL 0.64 | LM4 0.021 | SPL 0.64 | LM4 0.026 | LM4 0.59 | LM4 0.026 | PBS 0.57 | LM4 0.025 | PBS 0.57 | LM4 0.028 |
| | | SPL 0.65 | SPI 0.053 | PBS 0.63 | SPI 0.069 | PBS 0.62 | PBM 0.071 | PBS 0.57 | SPI 0.054 | LM4 0.55 | PBM 0.065 | LM4 0.52 | PBM 0.072 |
| | | LM4 0.64 | PBM 0.064 | LM4 0.63 | PBM 0.069 | LM4 0.59 | MOO 0.094 | STR 0.56 | MOO 0.056 | STR 0.53 | MOO 0.070 | PBZ 0.52 | MOO 0.078 |
| | | PBM 0.61 | MOO 0.080 | PBM 0.59 | MOO 0.081 | PBM 0.58 | HIL 0.096 | LM0 0.54 | PBM 0.061 | PBM 0.51 | SPI 0.071 | STR 0.51 | HIL 0.085 |
| | | LM0 0.61 | STR 0.089 | LM0 0.59 | HIL 0.088 | LM0 0.55 | SPI 0.105 | PBM 0.51 | SPL 0.067 | PBZ 0.50 | HIL 0.081 | PBM 0.50 | SPI 0.093 |
| | | GIT 0.58 | PBS 0.090 | PBZ 0.55 | PBS 0.092 | PBZ 0.54 | PBS 0.115 | PBZ 0.50 | STR 0.067 | LM0 0.49 | STR 0.084 | HIL 0.48 | SPL 0.099 |
| | | PBZ 0.57 | HIL 0.091 | GIT 0.54 | STR 0.095 | HIL 0.54 | SPL 0.125 | HIL 0.47 | HIL 0.067 | HIL 0.48 | PBS 0.088 | MOO 0.48 | PBS 0.101 |
| | | SPI 0.54 | SPL 0.092 | HIL 0.54 | SPL 0.095 | SPI 0.53 | STR 0.128 | MOO 0.46 | PBS 0.068 | MOO 0.47 | SPL 0.091 | LM0 0.46 | STR 0.104 |
| | | HIL 0.54 | PBZ 0.125 | SPI 0.52 | PBZ 0.131 | MOO 0.52 | PBZ 0.135 | GIT 0.46 | PBZ 0.093 | SPI 0.44 | PBZ 0.111 | SPI 0.45 | PBZ 0.122 |
| | | MOO 0.53 | APP 0.192 | MOO 0.52 | APP 0.177 | GIT 0.52 | APP 0.199 | SPI 0.44 | APP 0.102 | GIT 0.43 | APP 0.134 | GIT 0.41 | APP 0.135 |
| | | SND 0.29 | SQR 0.194 | SND 0.30 | SQR 0.197 | SND 0.31 | SQR 0.203 | SND 0.27 | SQR 0.125 | SND 0.27 | SQR 0.161 | SND 0.30 | SQR 0.162 |

**Figure 2.12** Relative ranking of treemapping algorithms for all data classes with less than four levels of hierarchy. Each table cell shows algorithms in top-down decreasing order of visual quality (left column) and stability (right column).



| | LWV | | | | | | HWV | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | RWC | | SWC | | LWC | | RWC | | SWC | |
| 4+L LID | LM4 0.65 | PBM 0.000 | | | | | APP 0.63 | SND 0.001 | | | | |
| | LM0 0.65 | GIT 0.000 | | | | | GIT 0.62 | LM0 0.002 | | | | |
| | GIT 0.65 | LM4 0.000 | | | | | LM4 0.60 | GIT 0.004 | | | | |
| | APP 0.65 | LM0 0.000 | | | | | LM0 0.59 | SPL 0.009 | | | | |
| | SPL 0.64 | SND 0.000 | | | | | SQR 0.57 | SPI 0.010 | | | | |
| | PBS 0.62 | SPL 0.001 | | | | | SPL 0.57 | PBS 0.010 | | | | |
| | STR 0.61 | SPI 0.001 | | | | | PBS 0.54 | LM4 0.012 | | | | |
| | SQR 0.58 | STR 0.002 | | | | | STR 0.47 | MOO 0.012 | | | | |
| | SPI 0.57 | HIL 0.003 | | | | | PBM 0.46 | PBZ 0.015 | | | | |
| | PBM 0.49 | PBS 0.003 | | | | | PBZ 0.43 | HIL 0.018 | | | | |
| | PBZ 0.41 | PBZ 0.004 | | | | | SPI 0.42 | STR 0.020 | | | | |
| | HIL 0.41 | MOO 0.004 | | | | | MOO 0.37 | APP 0.021 | | | | |
| | MOO 0.36 | APP 0.007 | | | | | HIL 0.37 | PBM 0.027 | | | | |
| | SND 0.04 | SQR 0.039 | | | | | SND 0.13 | SQR 0.050 | | | | |
| 4+L RID | | | SQR 0.74 | SND 0.009 | | | | | | | APP 0.66 | SND 0.006 |
| | | | STR 0.68 | LM0 0.011 | | | | | | | SPL 0.55 | GIT 0.008 |
| | | | APP 0.68 | LM4 0.011 | | | | | | | PBS 0.51 | LM0 0.008 |
| | | | SPL 0.64 | GIT 0.012 | | | | | | | LM4 0.50 | LM4 0.011 |
| | | | LM4 0.63 | SPL 0.049 | | | | | | | SQR 0.49 | MOO 0.040 |
| | | | PBS 0.62 | PBS 0.051 | | | | | | | PBZ 0.44 | STR 0.053 |
| | | | GIT 0.61 | STR 0.054 | | | | | | | GIT 0.42 | SPL 0.055 |
| | | | LM0 0.59 | SPI 0.065 | | | | | | | STR 0.40 | HIL 0.056 |
| | | | PBM 0.57 | MOO 0.079 | | | | | | | PBM 0.40 | PBS 0.059 |
| | | | PBZ 0.53 | HIL 0.087 | | | | | | | LM0 0.38 | SPI 0.066 |
| | | | SPI 0.50 | PBM 0.088 | | | | | | | MOO 0.36 | PBZ 0.067 |
| | | | HIL 0.45 | PBZ 0.140 | | | | | | | HIL 0.34 | APP 0.072 |
| | | | MOO 0.44 | SQR 0.161 | | | | | | | SPI 0.33 | PBM 0.099 |
| | | | SND 0.20 | APP 0.168 | | | | | | | SND 0.15 | SQR 0.125 |
| 4+L SID | SQR 0.69 | SND 0.003 | SQR 0.72 | SND 0.007 | APP 0.64 | GIT 0.013 | APP 0.64 | SND 0.011 | SQR 0.71 | SND 0.010 | APP 0.64 | SND 0.011 |
| | STR 0.69 | LM0 0.006 | APP 0.68 | LM0 0.010 | SQR 0.63 | SND 0.014 | SQR 0.58 | LM0 0.017 | APP 0.68 | LM0 0.016 | SQR 0.58 | LM0 0.017 |
| | PBS 0.63 | GIT 0.008 | STR 0.67 | GIT 0.011 | SPL 0.60 | MOO 0.047 | SPL 0.57 | GIT 0.017 | STR 0.63 | GIT 0.018 | SPL 0.57 | GIT 0.019 |
| | APP 0.63 | LM4 0.008 | SPL 0.64 | LM4 0.016 | LM4 0.59 | HIL 0.055 | LM4 0.56 | LM4 0.024 | SPL 0.62 | LM4 0.021 | PBS 0.53 | LM4 0.024 |
| | HIL 0.63 | SPI 0.010 | PBS 0.62 | SPL 0.052 | STR 0.59 | LM0 0.059 | PBS 0.53 | MOO 0.041 | PBS 0.60 | MOO 0.053 | LM4 0.53 | MOO 0.055 |
| | SPL 0.62 | MOO 0.013 | LM4 0.61 | MOO 0.058 | PBS 0.59 | LM4 0.063 | STR 0.50 | HIL 0.047 | LM4 0.55 | SPL 0.061 | STR 0.49 | HIL 0.064 |
| | MOO 0.61 | STR 0.017 | GIT 0.60 | PBS 0.060 | LM0 0.56 | PBS 0.069 | LM0 0.49 | SPL 0.049 | PBM 0.55 | PBS 0.064 | LM0 0.46 | PBS 0.064 |
| | PBM 0.60 | PBM 0.018 | LM0 0.58 | HIL 0.072 | PBM 0.55 | SPL 0.076 | PBM 0.46 | PBS 0.050 | PBZ 0.52 | HIL 0.069 | PBM 0.45 | SPL 0.065 |
| | LM4 0.60 | SPL 0.022 | PBM 0.57 | STR 0.079 | GIT 0.54 | PBM 0.079 | PBZ 0.45 | APP 0.051 | GIT 0.50 | PBZ 0.095 | PBZ 0.45 | STR 0.072 |
| | LM0 0.59 | PBS 0.023 | SPI 0.53 | PBZ 0.093 | SPI 0.51 | STR 0.094 | SPI 0.43 | STR 0.052 | LM0 0.50 | STR 0.097 | GIT 0.42 | SPI 0.073 |
| | GIT 0.56 | HIL 0.024 | PBZ 0.52 | PBM 0.094 | PBZ 0.50 | SPI 0.096 | GIT 0.40 | SPI 0.053 | SPI 0.50 | SPI 0.098 | SPI 0.42 | PBZ 0.080 |
| | PBZ 0.56 | PBZ 0.038 | MOO 0.45 | SPI 0.103 | HIL 0.48 | APP 0.128 | HIL 0.39 | PBZ 0.057 | HIL 0.46 | PBM 0.105 | HIL 0.40 | PBM 0.086 |
| | SPI 0.52 | APP 0.083 | HIL 0.43 | APP 0.129 | MOO 0.46 | PBZ 0.141 | MOO 0.37 | PBM 0.063 | MOO 0.46 | APP 0.124 | MOO 0.40 | APP 0.089 |
| | SND 0.22 | SQR 0.089 | SND 0.20 | SQR 0.161 | SND 0.24 | SQR 0.148 | SND 0.19 | SQR 0.070 | SND 0.18 | SQR 0.136 | SND 0.19 | SQR 0.109 |

**Figure 2.13** Relative ranking of treemapping algorithms for all data classes with four or more levels of hierarchy. Each table cell shows algorithms in top-down decreasing order of visual quality (left column) and stability (right column).

In general there are a number of insights we can obtain from Figures 2.12 and 2.13. When considering only the visual quality, SQR is usually the best for low weight variance data, but for high weight variance APP is just as often the best algorithm. If the dataset contains only 1 level, SQR performs better, but otherwise it depends on the exact data class. If only the stability is important, SND scores best on almost all data classes, but likewise it consistently scores the poorest on visual quality The state-aware algorithms all perform very well on stability. While LM0 is better in terms of stability than LM4, their exact order as well as their relative order to GIT varies depending on the data class.

## 2.7 Discussion and conclusion

In this chapter we investigated temporal coherence for time-varying hierarchical data. We first presented a new algorithm to compute stable treemaps for time-varying hierarchical data using the concept of local moves: small modifications to the treemap that influence only a small part of the treemap. These local moves allow us to control the trade-off between the visual quality and the stability simply by limiting the number of local moves between every two time steps. Moreover, in contrast to existing treemapping algorithms, the local moves allow for the full range of options to be explored when choosing layouts which can provably lead to treemaps with better visual quality.

We then performed an extensive quantitative evaluation of rectangular treemapping algorithms for time-varying data. To do so, we introduced a new methodology based on baseline treemaps to measure the stability of time-varying treemaps. Baseline treemaps enable us to measure the change in the input data in a manner that is mathematically comparable to the measures for the layout change of the corresponding treemaps. Furthermore, we proposed a novel classification scheme for time-varying datasets via a four-dimensional feature space (weight variance, weight change, tree depth, and the pattern of insertions and deletions). These four features naturally arose from a discussion on various types of state-of-the-art treemapping algorithms. Our experimental analysis shows that our proposed classification is valid in general and that most data classes are well suited to predict the performance of treemapping algorithms. For most data classes our visual summary comparing all algorithms across all data classes and both metrics can hence serve as a reliable resource for researchers and practitioners. Last but not least, all datasets, metrics, and algorithms used in our evaluation are openly available [137].

**Limitations and future work** The main disadvantage of our stable Local Moves algorithm is its running time which may be prohibitive for interactive applications on very large datasets. Nonetheless, for most reasonable practical scenarios where the treemaps still need to remain readable our algorithm is sufficiently fast. Beyond that, it is even possible to control the trade-off between the running time of the algorithm and the visual quality of the treemaps, again by controlling the number of local moves between every two time steps. A fully controllable trade-off between the three aspects visual quality, stability, and running time remains an interesting open problem.

Our experiments in the quantitative evaluation show that the features we identified and the resulting feature space generally work well and result in a meaningful classification of datasets. However, there are whole sets of data classes for which we could not find sufficiently many (or even any) datasets. This is partially inherent in the classification and somewhat natural: datasets with low weight variance hardly ever exhibit spiky weight change behavior, so that particular column in our table is essentially empty. But among the 18 classes of treemaps with 4 or more levels we found a significant number of datasets only for two classes which both are essentially populated by datasets stemming from software repositories. The question remains if there are other significant types of time-varying hierarchical datasets which have four or more levels and which escaped our searches. As it is, the results for these two particular classes are representative for only a restricted type of data.

Our classification works well for visual quality with the exception of two cases (2/3 level, spiky insertions and deletions, high weight variance, and low or spiky weight change). We have a large number and variety of datasets at our disposal for these two classes, but nevertheless, it is unclear to us what causes these inconsistencies in the performance of the tested algorithms. There might be a hidden correlation in these datasets and one or more additional features might be needed to separate these classes further.

While we do have a significant number of datasets at our disposal and hence can validate our claims with some certainty, we still might be observing some bias in our collection. As stated above, essentially all datasets with 4 or more levels stem from software repositories. Furthermore, all World Bank datasets have at most 3 levels. It would be interesting to analyze if and how this bias in the data influences our results. To overcome possible data bias we would also like to construct, and evaluate on, synthetic datasets. Doing so is not trivial; creating datasets that avoid sampling biases and are representative of real-world datasets (for a suitable definition of "real-world") is a challenging but important question in its own right in information visualization in particular and in data science in general.

Finally, our evaluation currently does not measure the running times and correspondingly the scalability of the algorithms used in our experiments. Our implementations are not (equally) optimized and hence a fair comparison is currently impossible. Scalability is clearly an important factor in online usage scenarios and we hope to be able to complement our current set of implementations with optimized versions in the near future.

# Chapter 3

# Uncertainty Treemaps

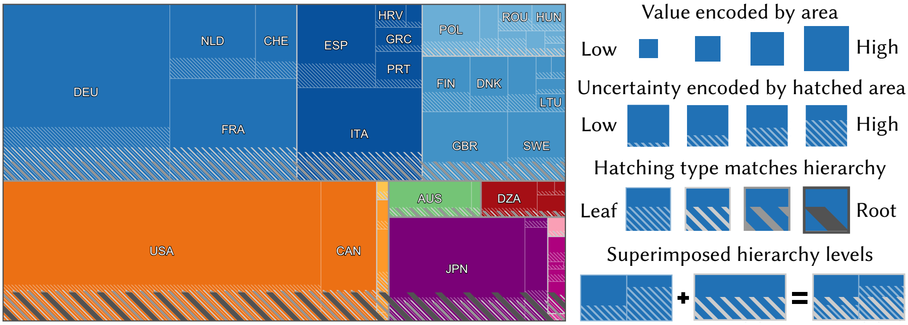


**Figure 3.1** Uncertainty Treemap of coffee import from 1994 to 2014, decomposed into continents, subregions, and countries [144]. Our visualization encodes uncertainty using nested hatched lines in the areas of the corresponding values. Of the North American countries (🟧), the United States (USA) import much more coffee than Canada (CAN). While Canada has much less uncertainty in its value (area of ▧), both countries share a similar relative fluctuation (height of ▧ relative to containing rectangles). Similarly, going up two levels (▨) reveals that Europe (🟦, 🟦, 🟦, 🟦) has lower relative fluctuation than the Americas (🟧, 🟧).

Numerical hierarchical data often has uncertainty associated with it which can present itself in many different forms, see Fisher [42] for an overview. Here, we focus on the uncertainty of the numerical values associated with each data element. When visualizing low complexity data it is often straightforward to indicate the error, for example, using error bars. Such visualizations anchor the uncertainty at visual representation of the data value. While this strategy works well for low-dimensional data, it does not lend itself directly to more complex or higher-dimensional data. In the context of treemaps an additional challenge is the fact that uncertainty does not aggregate in the same way as the data values do in the hierarchy: relative uncertainty tends to become smaller, the higher a node is in the hierarchy. Despite these challenges, any visualization that aims to be trustworthy and faithful to the data it represents has to visualize uncertainty for each node. Moreover, to ensure a coherent visualization of the direct relation between the value and its uncertainty, the uncertainty should be visualized together with the data. Nevertheless, techniques that support plotting certain and uncertain data simultaneously for more complex data types are rather scarce [62].

In this chapter we introduce a coherent treemap visualization for uncertain hierarchical data, focusing on rectangular treemaps. To do so, we identify two conflicting key requirements:

**Visual aggregation.** To assess the value of an interior node, the area of its rectangle (or generally, its visual size) should directly match its value.

**Uncertainty encoding by area.** To facilitate comparison between data and uncertainty, uncertainty should be encoded using the same visual variable as the data, that is, area.

Either requirement is straightforward to satisfy in isolation. For example, visual aggregation can be maintained if uncertainty is encoded using color or any other visual variable that does not hamper the perception of area. Similarly, uncertainty can be encoded using additional area adjacent to each rectangle. Doing so, however, will make visual aggregation next to impossible, as an inner node will have additional area inside its rectangle. In contrast to these decoupled approaches, our technique meets both requirements simultaneously, thereby improving coherence. As an integral part of our Uncertainty Treemaps we introduce *hierarchical uncertainty masks* as a tool to visualize data and uncertainty in the same visual space.

**Definitions and notation** In this chapter we use the following definitions and notation for treemaps; these differ slightly from those used in Chapter 2 as hierarchy plays a more central role in our Uncertainty Treemaps.

Let $\mathcal{T}$ be a rooted tree. Denote the $d$ children of a internal node $v \in \mathcal{T}$ with $C(v) = \{c_1, \dots, c_d\}$. A node without children is a leaf; non-leaf nodes are interior nodes. We denote the value of each node $v$ in $\mathcal{T}$ by $\mu(v)$ and we have $\mu(v) = \sum_{i=1}^{d} \mu(c_i)$ for any interior node $v$. We assume that $\mu$ corresponds to the mean and the sum of means, respectively. $\mu$ is normalized such that for the root node $v_r$, $\mu(v_r)$ equals the area of the input rectangle $R$. We denote the uncertainty of a node $v \in \mathcal{T}$ by $\sigma(v)$. Generally, the uncertainty of an interior node can be derived from its children. We assume that the uncertainty is the standard deviation and that the children are independent and thus $\sigma(v) = \sqrt{\sum_{i=1}^{d} \sigma(c_i)^2}$ for an interior node $v$. Hence, the relative uncertainty $\frac{\sigma(v)}{\mu(v)}$ tends to become smaller higher in the hierarchy.

We describe this layout using a function $R_\mu$ that maps each node $v \in \mathcal{T}$ to a rectangle with area $\mu(v)$. The rectangle for each interior node $v$ is the disjoint union of the rectangles of its children. The quality of a treemap is typically measured via the aspect ratio of the rectangles $R_\mu$ and denoted by $\rho(R_\mu(v))$ for a node $v$.

**Contributions and organization** Uncertainty Treemaps visualize the uncertainty in the same space as the data by using so-called *uncertainty masks*. These masks anchor a region $R_\sigma(v)$ of area $\sigma(v)$ to $R_\mu(v)$ for each node $v$. Since $R_\mu$ already covers the graphical space, we require that $R_\sigma(v) \subseteq R_\mu(v)$. Our design is based on screen-door transparency and renders the regions $R_\sigma(v)$ on top of the regions $R_\mu(v)$. In Section 3.2 we describe the design of our uncertainty masks in greater detail and explain how to overlay the masks hierarchically on the treemap layout $R_\mu$. We also show how to answer questions on real-world data using Uncertainty Treemaps. Furthermore, we introduce a new quality metric that measures the quality of a mask.

While our hierarchical uncertainty masks can be applied to any treemap layout computed with any treemapping algorithm, certain layouts result in better mask quality. To compute Uncertainty Treemaps with high mask quality, we could focus on this optimization criterion while ignoring the aspect ratio of the rectangles. However, the (summed) mean values still need to remain legible and comparable. Thus, in Section 3.3 we show how to adapt existing treemapping algorithms to take the uncertainty masks into account. Here, we distinguish two types of algorithms: *mask-friendly* algorithms that use only the fact that a mask will be placed and *mask-aware* algorithms that use the uncertainty values to compute the layout.

In Section 3.4 we experimentally compare variants of our mask-quality metric to establish how well these measures are able to capture different aspects of quality. Furthermore, we investigate the effectiveness of mask-friendly and mask-aware algorithms on several real-world datasets. Finally, in Section 3.5 we discuss various design alternatives for the shape, placement, and rendering of the masks, as well as the effects of uncertainty and unbalanced hierarchies.

## ▸ 3.1 Related work

In Chapter 2 we already gave an overview of state-of-the-art rectangular treemapping algorithms for time-varying data without uncertainty. In general, we may derive uncertainty from the temporal variation of a numerical value. Uncertainty Treemaps can thus straightforwardly visualize time-varying data with a static hierarchy, presenting a different visualization than the time-varying treemap visualizations in Chapter 2. Although this representation loses information about temporal patterns beyond the magnitude of change, Uncertainty Treemaps provide a simple, static overview visualization, removing the need for animation.

In this Section, we focus on related work for uncertainty visualization. Uncertainty visualization in general is a broad field, ranging from data acquisition to mapping to representation, and reasoning about uncertainty [22, 63]. We consider uncertainty visualization to be a process of multiplexing certain and uncertain data in such a way that humans can successfully demultiplex it [27, 61] and reason about it [49, 106]. Within this information-theoretical consideration we can distinguish between encoder, channel, and decoder. Correspondingly, an uncertainty-aware treemap has to have at least three channels, i.e., visual variables, to communicate tree structure, certain data, and uncertain data simultaneously. We map both data value and uncertainty to size, and use texture to separate value from uncertainty.

Bertin [12] pioneered the study of visual variables many years ago. However, the study of those variables in the context of uncertainty visualization is a fairly recent trend. Specifically, in the context of maps and graph edges, fuzziness appears to be an intuitive and sufficient choice [53, 79]. Clean geometry, e.g., boxplot whiskers, appears to be slightly less prone to error in general [51, 79]. However, there are some pitfalls such as unprofessional appearance [19] and the within-box bias [33]. These visual variables are not suitable for our technique, as we have to depict two types of data in one area, i.e., we have to find an uncertainty-aware blending operator.

Holliman *et al.* [61] recently coined the term *Visual Entropy*. Here, the idea is to couple high noise with an expected frequency to communicate uncertainty. Using white noise to depict uncertainty was evaluated and recommended in the context of maps [69]. Kale *et al.* [64] show that animation is also viable for framing uncertainty as frequency. To us, screen-door transparency seems particularly promising, since it avoids color blending issues without requiring additional space [66, 108]. Bair *et al.* [6] show that the number of fully overlapping terrain-like layers and choice of texture strongly influence the separability of those layers. Thus, our work adapts various types of screen-door transparency while preventing data from being fully concealed by the uncertainty, since partial overlap influences separability to a lesser degree.

Slingsby *et al.* [121] use a geospatial variant of treemaps in an interactive system to support the exploration of uncertainty in the context of geospatial data. In contrast to our Uncertainty Treemaps their approach does not anchor uncertainty to the data value, and relies on interaction to explore uncertainty. Most closely related to our work are the Bubble Treemaps proposed by Görtler *et al.* [48]—a variant of circular treemaps that can depict uncertainty using the contours of the regions, e.g., using modulated splines as the contour. Their approach separates the encoding of certain and uncertain aspects of data which hampers with comparison due to missing alignment and anchoring. Furthermore, Bubble Treemaps require ample white space and hence deliberately violate the visual aggregation requirement. See Figure 3.2 for a visual comparison between Uncertainty Treemaps and Bubble Treemaps. We argue that Uncertainty Treemaps provide a clearer and cleaner picture of the data, in particular, because our approach does not introduce unequally distributed whitespace to compensate for conflicting propagation models.

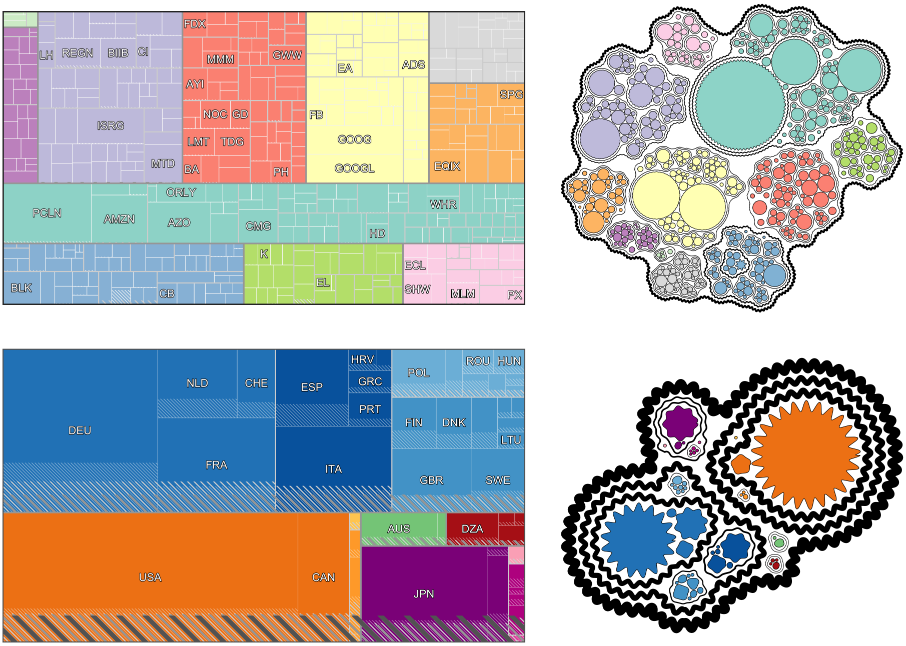


**Figure 3.2** Uncertainty Treemaps (left) and Bubble Treemaps [48] (right). For the S&P dataset [48] (top) the Uncertainty Treemap shows that this dataset has few data values with high uncertainty, whereas this is harder to establish for the Bubble Treemap. For the Coffee dataset [126] (bottom) the Bubble Treemap produces thick and folded boundaries that prohibit value comparison; in contrast, the Uncertainty Treemap indicates the aggregated uncertainty for each level in the hierarchy.

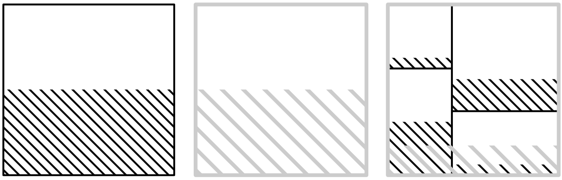


**Figure 3.3** Hatched uncertainty masks: a leaf node (left), a node above leaf level doubles line width and spacing (middle), a 2-level treemap with 4 leaves (right).

## 3.2 Hierarchical uncertainty masks

Our hierarchical uncertainty masks are key to enabling Uncertainty Treemaps in which $R_\sigma(v) \subseteq R_\mu(v)$ for any node $v$. Here, we first discuss the design of our uncertainty masks, then show how to read the resulting Uncertainty Treemaps, report the results of a brief expert review, and finally describe how we measure mask quality.

### 3.2.1 Mask design

Consider some (rectangular) treemap layout described by $R_\mu$. We want to augment the layout with regions dedicated to showing $\sigma$: the region $R_\sigma(v)$ of a node $v$ must be contained in $R_\mu(v)$ and have size $\sigma(v)$. Due to the recursive nature of treemaps, the region $R_\sigma(v)$ overlaps the rectangles (and masks) of nodes lower in the hierarchy. We must thus consider how to effectively render the masked regions to ensure that both $R_\mu$ and $R_\sigma$ remain visible.

In terms of placement, our mask spans a fraction of $\sigma(v)/\mu(v)$ of $R_\mu(v)$.[1] The mask could be essentially of arbitrary shape as long as it has the correct area as prescribed by $\sigma$. However, it is natural to use rectangles for the mask as well, since the mask is integrated into a rectangular treemap. In particular, we place the mask as a rectangle at the bottom of the node along the full width of the node; Figure 3.3 (left) and (middle) show masks for a single node.

Figure 3.3 also illustrates that we render the masks using hatching of slanted, parallel, equidistant lines. The line width and gap depend on the node's hierarchy level, both doubling with each higher level. The line gap is three times the line width such that at least every other line is fully visible. Figure 3.3 (right) illustrates the overlay of two masks at the bottom of the treemap: both the lower-level masks in $R_\sigma$ as well

---

[1] The observant reader will notice that for $\sigma(v) > \mu(v)$, this may be problematic. We refer to Section 3.5.3 for a brief discussion of this issue.

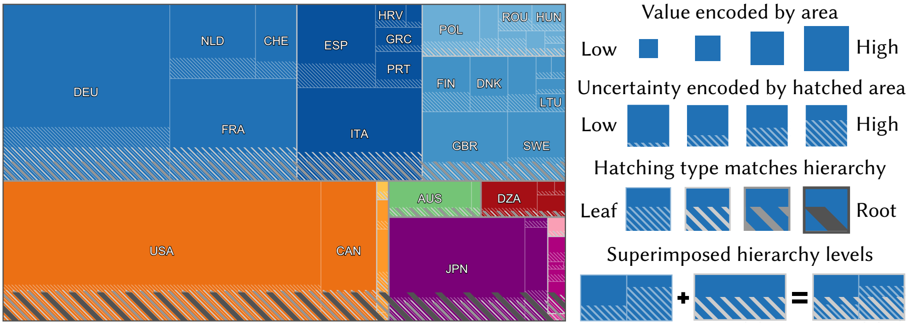


**Figure 3.4** Uncertainty Treemap of coffee import from 1994 to 2014, decomposed into continents, subregions, and countries [144].

as the full extent of rectangles in $R_\mu$ remain visible. Note that, contrary to other approaches, the pattern density does *not* encode the uncertainty value but matches the hierarchy level.

We place the hatching such that mask lines at a certain level coincide with half of the mask lines one level lower. To this end, we define the hatching pattern globally within $R$. The mask for a specific node $v$ is then the intersection of $R_\sigma(v)$ with the global mask which ensures that the intended visibility through the masks is maintained regardless of the positioning of the nodes. A discussion of alternative mask shapes and patterns can be found in Section 3.5.

### 3.2.2 Reading an uncertainty treemap

Figure 3.4 again shows an Uncertainty Treemap of the Coffee dataset [126]: the mean amount of coffee imported yearly per country between 1994 and 2014 with the associated standard deviation [144]. We illustrate our technique by answering several example questions using Figure 3.4.

*Compare the United States (USA) and Canada (CAN): which country imports the most coffee?* To perform this standard task for a treemap, we compare the areas of USA and CAN. In this particular case the rectangles of the USA and CAN have the same height: we can hence infer immediately that the rectangle for the USA spans a larger area and thus the USA import more coffee.

*Compare the USA and CAN: which country shows the most fluctuation in import?* To compare the absolute standard deviation, we compare the hatched area (▨) of the

USA with that of CAN. Clearly, the area of the USA is larger than that of CAN: it has considerably more variation. Note that we cannot just rely on the height of these rectangles here as their width is different.

As the mean and standard deviation are both visualized using area, we can actually also consider this question in relative terms. Both hatched areas fill approximately a third of the corresponding rectangle—this can easily be estimated by just considering the height, since the hatched area and the rectangle span the same width. Since the rectangles have the same top and bottom boundary, we can even make a fairly accurate decision: USA have higher relative fluctuation than CAN because its hatched area ends slightly higher.

*Consider the questions above, now comparing Europe (EU🟦, 🟦, 🟦, 🟦) with North America (NA 🟧).* As the rectangles for EU and NA have different width and height, we cannot take shortcuts like above, and must use area to answer questions regarding the summed mean and standard deviation of the import. As there is significantly more blue area (🟦, 🟦, 🟦, 🟦) than dark orange area (🟧), Europe imports more coffee.

For assessing absolute fluctuation for EU, we look at the hatching at continent level (▨) which here spans the entire width of the treemap. NA is one level lower in the hierarchy, and thus we consider the subcontinent level hatching (▨). These areas show slightly higher fluctuation in NA despite EU having a larger mean.

In relative terms, we consider the same hatched areas but now with respect to their surrounding rectangle. We see that EU has approximate 15% relative fluctuation whereas NA has about a fourth: NA has higher relative fluctuation. Note that we can always rely purely on height of the hatched areas and their containing rectangle to infer the relative fluctuation for any node.

### 3.2.3 Expert review

To investigate whether our uncertainty masks yield usable treemaps visualizations, we conducted an informal review of our method with a visualization and perception expert. Below, we describe the setup, results, and conclusions of this review session. To keep things simple, we positioned the expert review for visualizing uncertainty, but restricted terminology in questions to mean values and standard deviations.

**The expert** The review was done with a visualization and perception expert. He was not involved with or informed of our new method before the review session, but is generally knowledgeable about and familiar with treemaps.

**Techniques** To avoid scoping our review too narrowly, we used three techniques. In addition to Uncertainty Treemaps and Bubble Treemaps [48], we used a simple Juxtaposed variant. The latter shows a standard treemap on the mean values side-by-side with a treemap on the uncertainty values. The treemap on the uncertainty values uses the same combinatorial layout as the treemap on the mean values. We generate this treemap by running the algorithm and making any decisions based on the mean, but eventually visualizing the uncertainty. Although this may be suboptimal in terms of aspect ratio, it should be easier to visually relate the two treemaps. Before the review, we judged that the layouts showing uncertainty had visual quality comparable to using a regular treemapping algorithm on the uncertainty data. Note that in the Juxtaposed variant only the leaves show correct values. Internal nodes are typically larger than their actual uncertainty.

**Tasks** To familiarize the expert with the techniques and uncover potential pitfalls or misconceptions we asked the expert to complete a few tasks with each of the visualizations. Per visualization we asked 6 questions, each with a variant on leaf level (a) and on interior nodes (b).

Question 1 and Question 2 ask to compare the mean value and standard deviation respectively of two given nodes. Question 3 asks to estimate the ratio between the standard deviation and the mean value of a single given node (which we refer to here simply as “ratio”). Question 4 through 6 reflect these same questions, but instead ask to find the node with the highest mean, standard deviation, or ratio.

**Datasets** We used the “Coffee” dataset as Dataset 1 for Questions 1 through 3 (comparisons) for all techniques, using different countries. For Questions 4 through 6, we used the “Infant” dataset as Dataset 2; we perturbed the values for this dataset between the visualization techniques to avoid the answers necessarily being the same. The expert was explained that these datasets represent countries aggregated into regions or subcontinents, subsequently aggregated into continents.

**General procedure** All material was printed separately full size on a sheet of A4 paper. First, the general setting was introduced of visualizing a hierarchy with mean values and standard deviations simultaneously. We explained the general setup to the expert and he was requested to think aloud and ask questions as they arose. We would occasionally ask clarifying questions about how certain answers were obtained to discover whether the applied process for reading and finding the answers matches with how the techniques are intended to be used.

The questions for Dataset 1 were provided first, treating the techniques in the order of Juxtaposed, followed by Bubble Treemap, followed by Uncertainty Treemap. For each technique, the expert was first introduced to the visualization, purposefully limited to the basic encoding used, without explaining in detail how a question was to be answered. Next, the expert answered the three questions before starting with the next technique. Subsequently, the questions from Dataset 2 were provided, following the same procedure for Dataset 1, without the explanation of the techniques. As not all countries had labels, the expert was also given the option to mark his answers directly on the visualization; this was used only to answer all Bubble-Treemap questions. After Dataset 2, we asked the expert about things that stood out to him when completing the tasks, thoughts about the usability of the techniques, and his preferences. After indicating his preferences we revisited some limitations of the techniques. Below, we structure the comments and findings per technique during the tasks before reviewing the results of the final discussion.

**Completing tasks** The Juxtaposed method was straightforward to use and did not prompt any questions. The expert did remark that finding the maximum ratio (Question 6) “is a tough one”[2] as it requires linking between two side-by-side figures.

For Bubble Treemaps, the expert remarked that there was no frame of reference (legend) for the visualized uncertainty instead relying on “gut feeling” to estimate a ratio. For assessing area of interior nodes the expert found that this is “very hard to do”, as “there is a lot of white space in between, severely distorting what you see and this is exacerbated by the contour waves”.

For Uncertainty Treemaps, the expert completed the tasks without questions or remarks. When prompted how he answered the ratio question, he explained that he used the height of the hatched area to assess how often the hatched area would fit into the height of the containing rectangle. Though the initial explanation instructed only in area, this shortcut seemed evident. However, a later question involved comparing the standard deviation where height was interpreted as immediately encoding standard deviation. The expert indicated that he was “distracted by height”. This was clarified to answer subsequent questions, but was revisited in the discussion later.

**Preferences** For Questions 1, 2, 4, 5 – requiring to consider only means or only standard deviations – the expert indicated a slight preference for Juxtaposed over Uncertainty Treemaps: they are “essentially identical” because one is “not bothered

---

[2] All quotes are paraphrased as the review session was conducted in Dutch.

by the hatching". However, for Questions 3 and 6 – requiring to assess ratio – the expert had a clear preference for Uncertainty Treemaps. Bubble Treemaps was consistently the least preferred method.

**Discussion with the expert** There was an additional in-depth discussion on the design of Uncertainty Treemaps and specifically the masks with the expert. The initial question by the expert was why we opted for hatching rather than solid rectangles. He agreed with our rationale, that transparency would lead to color identification issues and placing the standard deviation next to the mean would cause similar problems with assessing aggregate area as he faced with Bubble Treemaps.

He identified the hatching as being such that one "is tempted to look at the length of the (individual) lines, and not interpret them as a holistic thing". He did think that increasing the density and decreasing the width may reduce this problem. Furthermore, coloring them more similarly to the background color could help in interpreting these as a holistic object and simultaneously reduce the visual prominence of the masks. When prompted about potential coloring issues for higher level nodes (with different background colors), he indicated that such colors could be expected to remain a shade of gray. He also mentioned that treemaps "take some time to get used to", and thus a learning curve is to be expected for any overlay on a treemap.

**Reflection** The expert review suggests that Uncertainty Treemaps are usable and potentially preferable over the two other methods for the evaluated tasks. The main issue identified is that the hatching may trigger height assessment even when this is not appropriate. This can be attributed to a learning curve—which is to be expected for any treemap representation according to the expert. We changed the rendering of the hatching to reduce such risks following the above discussion. Specifically, we now blend the white leaf-level hatching with the leaf-level coloring and doubled the pattern density. This indeed reduces the visual prominence of the masks while increasing the sense of area. All figures and descriptions in this chapter reflect the updated design.

We also tried applying this to the subcontinent-level hatching as the countries within the same subcontinent are given the same color. However, this causes slightly different colors at boundaries where subcontinents of the same continent meet following our design that hatching coloring is to match the boundary coloring. This increases the potential for misreading the hierarchy and thus we did not apply this. In general, we would thus recommend that such color blending is applied only at nodes that have the same color as their siblings.

Technically speaking, some questions were impossible to answer. Specifically, there is not enough information to answer Question 3 in a Bubble Treemap or Juxtaposed visualization without a way of linking the two scales applied. This, however, does not prevent some basic intuitive assessment which may thus be quite far off. Indeed, after working with the Uncertainty Treemap, the expert indicated that he would now revise his answers to Question 3 for the other techniques. Note that, though assessing the ratio directly is limited for these two techniques, making comparisons between ratios (Question 6) is in fact possible.

### 3.2.4 Mask quality

Hierarchical uncertainty masks can be applied to any treemap layout. However, the layout has a large effect on the readability of the masks, as the mask of an interior node is rendered on top of its descendants. Though aspect ratio is a prominent measure for assessing treemap layouts, our masks are immediately derived from such a layout and relative uncertainty can be derived from height only. Therefore, we introduce a new measure for mask quality of a given layout to enable comparison and optimization that more closely ties to the hierarchical nature of the masks.

Generally, we prefer the mask of a node to overlap few of its children to increase the visibility of the individual child nodes. However, mask overlap cannot be avoided due to the partitioning nature of treemaps.

When a child's mask extends beyond its parent's mask, it is more clearly visible as no mask is rendered on top of it and thus the number of layers that a reader must see through to assess the area is reduced. This has been shown to be beneficial for the separability of layered terrain-like surfaces [6]. Extending the mask also matches the fact that relative uncertainty decreases higher in the hierarchy. Our metric hence captures how much the parent's mask extends beyond the mask of its children.

**Node quality** To capture this idea for a single node $u$, we define its *excess overlap* with respect to its parent or ancestor $v$. The excess overlap that $v$ causes for $u$ is the part of the rectangle of $u$ that is covered by the mask of $v$ but not by the mask of $u$ itself: $\varepsilon(u, v) = (R_\mu(u) \cap R_\sigma(v)) \setminus R_\sigma(u)$ (see Figure 3.5). Let $S(u)$ be the set of the relevant ancestors of $u$. We define the excess overlap of $u$ as $\mathrm{EO}(u) = \sum_{v \in S(u)} |\varepsilon(u, v)|$. We consider two options for the set $S(u)$: either it contains all ancestors of $u$ (option A) or it contains only the parent of $u$ (option P). Although option P is less complex it may miss repeated excess overlap by many ancestors.

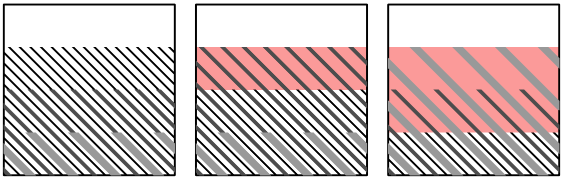


**Figure 3.5** Examples of mask quality, with excess overlap indicated by the shaded region. Properly nested: each lower-level mask (thinner lines) extends beyond the higher-level masks (left). The middle-level mask extends beyond the lowest-level mask (middle). Each higher-level mask extends beyond the lower level masks (right).

**Treemap quality** We now derive a mask-quality measure for an entire layout using excess overlap. There are multiple options for aggregating the values of individual nodes. The excess overlap measures the area of overlap, typically putting more focus on large nodes. We can either choose to directly aggregate the area or size (option S) of the excess overlap of individual nodes or to first normalize (option N) the excess overlap of each node $u$ by dividing $EO(u)$ by $\mu(u)$. Although option S captures visually salient excess overlap, it could result in hiding that many smaller nodes have excess overlap. If each node is equally important, regardless of area, then option N is more adequate to capture the overall visibility of the masks.

The various options lead to four different quality measures for an individual node $u$ which we denote by $\mathbf{EO}\boldsymbol{xy}(\boldsymbol{u})$ where $x \in \{\text{A}, \text{P}\}$ and $y \in \{\text{S}, \text{N}\}$. For example, EOAS($u$) indicates the excess overlap of $u$ measured with respect to all ancestors (A) of $u$ using the size (S) of the overlap directly; EOPN($u$) indicates the excess overlap of $u$ measured only with respect to the parent (P) of $u$ normalized (N) by the area $\mu(u)$ of $u$. Typically, we drop the argument $u$ when we talk about the quality measure in general.

Using these measures we can compute mask quality for an entire layout by taking the average or the maximum over all nodes in the hierarchy. In Section 3.4 we study the correlation between these measures for rectangular treemaps. Note that these quality measures can be directly applied to any type of treemap and mask shape.

## 3.3 Algorithms for uncertainty treemaps

There exist many different treemapping algorithms for hierarchical data without uncertainty as discussed in Chapter 2. Typically, they focus on optimizing the aspect ratio of the individual rectangles in the layout. When overlaying hierarchical uncertainty masks, we should also consider mask quality (Section 3.2.4). However, the primary data values need to remain legible and comparable and thus we cannot focus solely on mask quality; indeed aspect ratio for the rectangles remains a primary concern. The resulting problem is thus a bicriteria optimization problem: we want to optimize the aspect ratios but additionally achieve high quality of the uncertainty masks. We therefore adapt existing treemapping algorithms to take the uncertainty masks into account. Treemapping algorithms often include choices that do not influence the final aspect ratios of the resulting treemap. We aim to modify such choices to improve the quality of the masks. We can distinguish two types of mask-based modifications to the algorithms:

**Mask-friendly:** the algorithm uses the fact that the mask will be placed at the bottom of a rectangle but does not consider the actual uncertainty values.
**Mask-aware:** the algorithm uses also the uncertainty to compute the layout.

The degree to which we can make a particular treemapping algorithm mask-friendly or mask-aware depends on how many choices are arbitrary regarding the aspect ratios and thus how much freedom the algorithm has. We illustrate our approach with the algorithm that arguably has the most freedom: the Approximation algorithm.

**Approximation** The Approximation algorithm [91] works as follows. Consider a node $v$ with a given rectangle $R_\mu(v)$. Let $c_1, \dots c_d$ denote the $d$ children of $v$ sorted in decreasing order by value. Let $k$ denote the smallest number such that $\sum_{i=1}^{k} \mu(c_i) \geq \frac{1}{3}\mu(v)$. Let $A = \{c_1, \dots, c_k\}$ and $B = \{c_{k+1}, \dots, c_d\}$ denote the resulting partition. We now split $R_\mu(v)$ into two rectangular containers of the correct size: one for $A$ and one for $B$. If the width of $R_\mu(v)$ exceeds its height, this split is *horizontal*: the two containers are side-by-side, where $A$ is assigned to the left container and $B$ to the right. Otherwise, the split is *vertical*: one container is above the other, with the upper one being assigned to $A$ and the lower to $B$. This strategy is then recursively applied to the two containers, where we may now consider the container as a "virtual node" with the assigned children. When the container has only a single child $c$ it defines $R_\mu(c)$ and we may continue the recursion on $u$ if it is not a leaf. It can be shown that, using this approach, all aspect ratios are bounded by $\max\{\rho(R_\mu(v)), 3, 1 + \max_{1 \leq i < d} \mu(c_{i+1})/\mu(c_i)\}$ [91].

### 3.3.1 Mask-friendly algorithms

Mirroring a given treemap horizontally or vertically (or any of the rectangles of interior nodes) does not affect the aspect ratios of the rectangles in the treemap. However, it can affect the quality of the uncertainty mask. Some algorithms construct their layout based on the order of the children of a given node by area/value, typically placing the rectangles in a left-to-right and/or top-to-bottom pattern. Such algorithms naturally place children with small area at the bottom. However, since the uncertainty mask is drawn at the bottom of rectangles, the mask will typically overlap with the smaller children resulting in a lower-quality mask.

We can avoid this problem by mirroring the construction vertically, placing large children at the bottom and small children at the top. This simple approach can be applied to many different treemapping algorithms, making them mask-friendly. As shown by our experiments (Section 3.4) the approach generally improves mask quality.

We illustrate this approach explicitly for the Approximation algorithm. When the original algorithm intends to make a vertical split, then the larger children (set $A$) are placed above the smaller children (set $B$). We can easily mirror this approach by placing $B$ above $A$ instead without affecting aspect ratio. For a horizontal split, we do not need to change anything. The pseudocode for this mask-friendly version of the Approximation algorithm can be found in Algorithm 3 and Algorithm 4. The only change to the original algorithm is on line 7 of Algorithm 3 where “below” is used instead of “above”. Given a tree $T$ with root $v$ and input rectangle $R$ we compute a layout by calling MFApproximation($C(v), R$).

**Algorithm 3** MFApproximation($U, R$)

```
 1: if $d = 1$ then
 2:     $c \leftarrow$ the single node in $U$
 3:     MFApproximation($C(c), R_\mu(c)$)
 4: else
 5:     $s, A, B \leftarrow$ ApproximationSplit($U, R$)
 6:     if $s$ = vertical then
 7:         $R_A, R_B \leftarrow$ vertical split of $R$ with $R_A$ below $R_B$
 8:     else
 9:         $R_A, R_B \leftarrow$ horizontal split of $R$ with $R_A$ left of $R_B$
10:     MFApproximation($A, R_A$)
11:     MFApproximation($B, R_B$)
```

**Algorithm 4** ApproximationSplit($U, R$)

```
Sort $U = \{c_1, \dots, c_d\}$ by decreasing $\mu$ value
$k \longleftarrow$ smallest value such that $\sum_{i=1}^{k} \mu(c_i) \geq \frac{1}{3} \sum_{i=1}^{d} \mu(c_i)$
$A, B \longleftarrow \{c_1, \dots, c_k\}, \{c_{k+1}, \dots, c_d\}$
if height of $R$ is greater than its width then
    return vertical, $A, B$
else
    return horizontal, $A, B$
```

### 3.3.2 Mask-aware algorithms

The ability of mask-aware algorithms to use also the uncertainty opens up many possibilities to optimize mask quality. It allows us to directly use one of the quality measures for masks (Section 3.2.4) to guide the construction of the layout. However, optimizing for mask quality is difficult and likely cannot be computed efficiently. Nonetheless, we can use a mask-quality measure to make more informed heuristic choices in the treemapping algorithm.

We illustrate this approach by adapting again the Approximation algorithm to make a mask-aware variant: we explore the choices we can make in the algorithm further. The main choice involves the partitioning of the children $c_1, \dots, c_d$ (ordered decreasingly by $\mu$) of a node $v$ into $A$ and $B$. We recall that the aspect ratio of the final rectangles in the treemap is bounded by $\max\{\rho(R_\mu(v)), 3, 1 + \max_{1 \leq i < d} \mu(c_{i+1})/\mu(c_i)\}$. Thus, as long as the same bound applies to the subsets of children $A$ and $B$ (in particular, the ratios of consecutive values) and the aspect ratios of their container rectangles, the resulting partition will satisfy the aspect-ratio bound regardless of the choice of $A$ and $B$. We call a split (that is, the sets $A$ and $B$ along with the splitting direction) *valid* if it satisfies these properties. By allowing all valid splits we have more options to optimize mask quality. To increase flexibility even further, we introduce a parameter $q \geq 3$, and relax the aspect ratio bound to $\max\{\rho(R_\mu(v)), q, 1 + \max_{1 \leq i < d} \mu(c_{i+1})/\mu(c_i)\}$.

We now describe the modifications in more detail. The overall algorithm remains the same, but we replace the splitting algorithm ApproximationSplit by MaskAwareSplit (Algorithm 5), which attempts to find a valid split that optimizes mask quality.

**Choosing splits** As horizontal splits always cause both container rectangles to overlap the mask of $v$ we prefer vertical splits over horizontal splits. With a vertical split, the mask of $v$ may overlap rectangles and masks in the bottom container. However, if the bottom container is large enough, the mask of $v$ will not overlap the top container. Thus, we first attempt to find a valid vertical split and use a horizontal split only if no valid vertical split exists.

**Algorithm 5** MaskAwareSplit($U, R$)

```
Sort $U = \{c_1, \dots, c_d\}$ by decreasing $\mu$ value
$k, k' \leftarrow$ smallest and largest value such that $\{c_1, \dots, c_k\}$, $\{c_{k+1}, \dots, c_d\}$ is a valid vertical split
if $k$ and $k'$ exist then {*a valid vertical split exists*}
    $A, B \leftarrow \{c_1, \dots, c_k\}, \{c_{k+1}, \dots, c_d\}$
    $A', B' \leftarrow \{c_{k'+1}, \dots, c_d\}, \{c_1, \dots, c_{k'}\}$
    Improve estimate $e$ ($e'$) for split $A, B$ ($A', B'$) by repeatedly moving the node from $B$ ($B'$) to $A$ ($A'$) that most improves the estimate and maintains validity, until no such element exists
    if $e \leq e'$ then
        return vertical, $A, B$
    else
        return vertical, $A', B'$
else
    $k \leftarrow$ smallest value such that $\{c_1, \dots, c_k\}, \{c_{k+1}, \dots, c_d\}$ is a valid horizontal split
    $A, B \leftarrow \{c_1, \dots, c_k\}, \{c_{k+1}, \dots, c_d\}$
    Reduce imbalance $|\sum_{a \in A} \sigma(a) - \sum_{b \in B} \sigma(b)|$ for split $A, B$ by repeatedly moving the node from $B$ to $A$ that most reduces the imbalance while maintaining validity, until no such element exists
    return horizontal, $A, B$
```

**Vertical splits** As before, let $c_1, \dots, c_k$ be the children of a node $v$ in decreasing order on the $\mu$ value. We initially consider a partition $A = \{c_1, \dots, c_k\}$ and $B = \{c_{k+1}, \dots, c_d\}$, such that $A$ and $B$ form a valid vertical split, and $k$ is as small as possible. If no valid split exists we use a horizontal split instead. $A$ will be assigned to the lower container and $B$ to the upper container for a vertical split. Thus, we want to achieve a high mask quality (measured as a low score in one of our measures) for $A$ in the lower container. Recursively computing the optimal mask quality for $A$ would be too expensive. Instead, we estimate mask quality using a simple layout for $A$: all remaining splits in $A$ are horizontal. This way we can efficiently compute an estimate for any mask-quality measure (e.g., EOPN) in $A$.

Having found a valid vertical split we then use a local search algorithm to further improve the estimated mask quality in $A$. We repeatedly pick the node $b \in B$ such that: (1) moving $b$ from $B$ to $A$ still results in a valid vertical split, and (2) the estimated mask quality in $A$ (after moving $b$ to $A$) is improved the most. We repeat this step until no further improvement is possible.

Finally, we also try the same approach with the order of the children reversed. That is, we choose the largest $k'$ such that $A' = \{c_{k'+1}, ..., c_d\}$ and $B' = \{c_1, ..., c_{k'}\}$ is a valid vertical split. That is, $A'$ contains the smaller children and $B'$ the larger children. Note that this is different from the mask-friendly strategy, placing smaller children at the bottom. However, since the mask-aware algorithm uses the uncertainty values explicitly and small children may have large uncertainty, this approach can sometimes improve mask quality.

**Horizontal splits** If we apply a horizontal split, then the split into $A$ and $B$ does not influence the mask-quality estimation directly. However, for both $A$ and $B$ it is beneficial for mask quality to put elements with large uncertainty at the bottom in future vertical splits. Hence, we aim to make the total uncertainty in $A$ and $B$ as equal as possible.

As above, we start with a partition $A = \{c_1, \ldots, c_k\}$ and $B = \{c_{k+1}, \ldots, c_d\}$, such that $A$ and $B$ form a valid horizontal split, and $k$ is as small as possible. Next, we aim to minimize $|\sum_{a \in A} \sigma(a) - \sum_{b \in B} \sigma(b)|$. We again try to minimize this difference via a local search: we repeatedly move the node $b \in B$ from $B$ to $A$ that reduces the difference as much as possible while ensuring that the horizontal split remains valid. Since $A$ and $B$ are essentially symmetric for horizontal splits, there is no need to evaluate the reverse order of the children like we did for the vertical splits.

## 3.4 Experimental results

We use computational experiments to relate our quality measures (Section 3.4.1) and investigate the effectiveness of mask-friendly and mask-aware algorithms (Section 3.4.2 and 3.4.3, respectively).

**General setup** We consider the following six original algorithms in our experiments: the Approximation algorithm (**APP** [91]; the Split algorithm (**SPL**) [37]; the Squarified algorithm (**SQR**) [23]; the Strip algorithm (**STR**) [9]. SQR and STR are also implemented with a look-ahead [9], denoted by **SQRL** and **STRL**, respectively. We use the indicated abbreviations to refer to the original algorithm, and append **-F** to indicate mask-friendly implementations.

For our parameter $q$ (see Section 3.3) we indicate our mask-aware algorithm using **APP-$q$N** and **APP-$q$S** depending on whether we base the estimate on EOPN or EOPS. We set $q$ to 3 or 5, thus resulting in four mask-aware algorithms.

We run each of these 16 algorithms on the four real-world datasets below, providing a rectangle of width 1920 and height 1080 as the input rectangle. We measure the excess overlap according to the schemes described in Section 3.2.4. We do not investigate the running time in detail as each algorithm is effectively instantaneous on the datasets (less than 1 millisecond on a normal laptop). All datasets and algorithms are available online [143].

**Coffee:** mean amount of coffee import per country from 1994 to 2014 with the associated standard deviation [144].

**Infant:** mean number of infant deaths per country from 1992 to 2016 with the associated standard deviation [159].

**S&P:** mean closing price per stock in the Standard & Poor's 500 Index from 03-11-2016 to 10-11-2016 with the associated standard deviation.

**CES:** mean expenditure per (consumer) household in 2014; standard deviation represents the measurement uncertainty [15].

### 3.4.1 Relating mask-quality metrics

We introduced four variants of our mask-quality measure, depending on whether we measure excess overlap to the parent or all ancestors, and whether or not we normalize to fractions or measure based on actual area. We investigate here how different these measures are in capturing aspects of quality. To this end, we measure the correlation coefficient ($R^2$ score for the linear regression) between each of the two measures based on the quality of all dataset-algorithm pairs.

The choice between parent and all ancestors is highly correlated as shown in Figure 3.6 (left and middle). This high correlation implies that, at least on our datasets, the choice is not distinctive. In other words, the overhead of considering all ancestors does not change the effective quality ranks or differences significantly. Thus, we consider only excess overlap measured with respect to the parent in the remainder of this section.

The normalized variant and size variant (average EOPN and EOPS) are not as strongly correlated. The correlation is around 0.39 and Figure 3.6 (right) shows clear outliers. As such, we may consider these measures distinct. The size variant captures the amount of screen space (roughly matching visual saliency) of excess overlap but

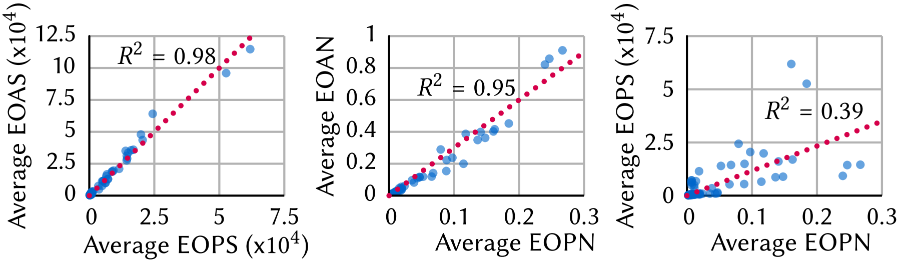


**Figure 3.6** Average EOAS and EOPS (left) and average EOAN and EOPN (middle) show a clear correlation. Average EOPN and EOPS do not show a clear correlation and thus capture mask quality differently.

may underestimate small nodes with low quality; the normalized variant treats all nodes equally, independent of their size.

### 3.4.2 Effect of mask-friendly algorithms

We implemented six algorithms, both in the original and in a mask-friendly manner (see Section 3.3.1). By design, mask-friendly implementations keep the same quality in terms of aspect ratio but may have a considerable impact on mask quality, essentially without cost. Here, we investigate this impact: we consider the average and maximum excess overlap of the nodes in the layout averaged over all datasets, and measured for both the original algorithms as well as the mask-friendly implementations. Figure 3.7 shows the results. Both in terms of average and maximum excess overlap, we see considerable improvement for most algorithms, but specifically for

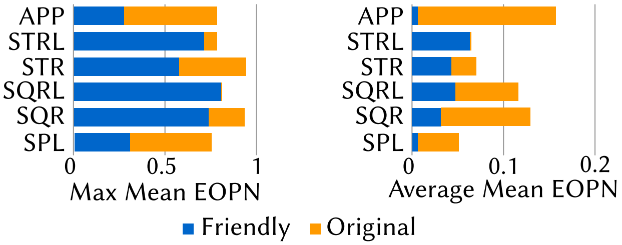


**Figure 3.7** Improvement in mask quality for mask-friendly algorithms compared to original implementations, based on maximum (left) and average (right) excess overlap. Mean is taken over all datasets.

APP and SPL. The notable exception is STRL, which is almost unaffected. For the EOPS metric, similar yet slightly stronger improvements are made.

Even though excess overlap is reduced using a mask-friendly approach, there is still ample room for improvement. Consider the Uncertainty Treemap of APP-F on the Infant dataset shown in Figure 3.8 (right-top). Though most nodes have little to no excess overlap, nodes in the bottom-most row after recursion do have excess overlap from their ancestors reducing their mask visibility.

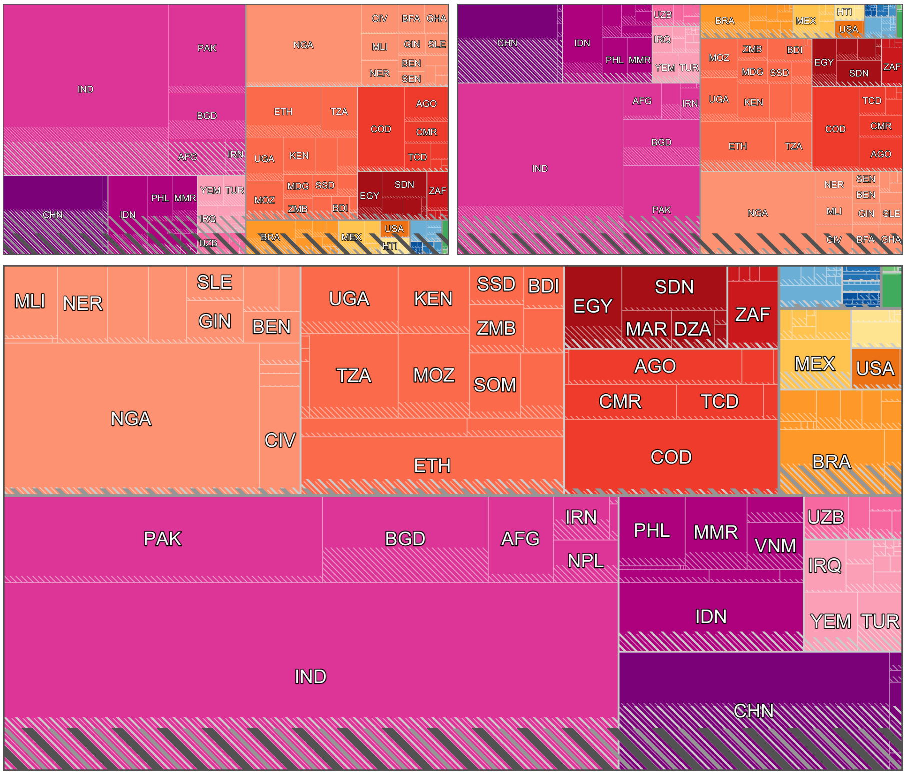


**Figure 3.8** Layouts of the Infant dataset computed by APP (left-top), APP-F (right-top) and APP-3N (bottom). The number of nodes with excess overlap, and the general amount of excess overlap decreases considerably with each improvement.

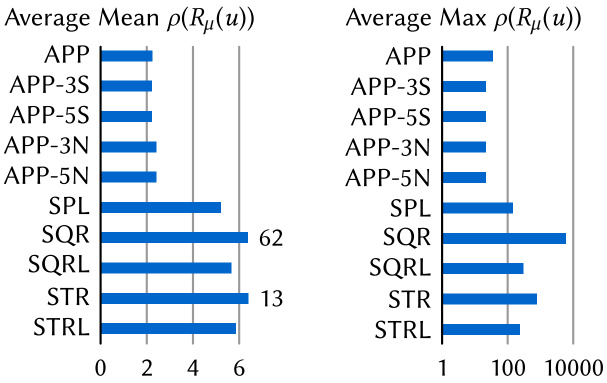


**Figure 3.9** Mean and maximum aspect ratio of all leaves in the layout, averaged over the datasets, for original and mask-aware algorithms. By their nature, original and mask-friendly algorithms achieve the same aspect ratio. Note the log-scale for the maximum.

### 3.4.3 Effect of mask-aware algorithms

We now consider our mask-aware algorithms, APP-**. The primary motivation for mask-awareness is to further improve mask quality by reducing excess overlap. Here, we investigate whether we can indeed observe such improvements compared to mask-friendly implementations, and which mask-aware variant performs best.

**Aspect ratio** Unlike mask-friendly algorithms, mask-aware algorithms may structurally change the layout such that the aspect ratios change. Hence, we first consider the effect of our modifications upon the aspect ratio of the rectangles.

Figure 3.9 shows the aspect ratio achieved by the algorithms. Compared to other algorithms, APP variants achieve considerably better ratio both in terms of the mean and the maximum. As we also concluded that APP-F achieves the best mask quality, we focus now exclusively on the mask-friendly and mask-aware variants of APP.

Between these APP variants we see comparatively little difference, but this also depends on the dataset. Coffee and CES datasets see a slight improvement in mean aspect ratio with mask-aware algorithms, but for the Infant and S&P datasets it decreases slightly. The resulting average remains approximately the same. In contrast, the maximum aspect ratio decreases drastically for CES and Coffee although there is also a slight increase for Infant and S&P.

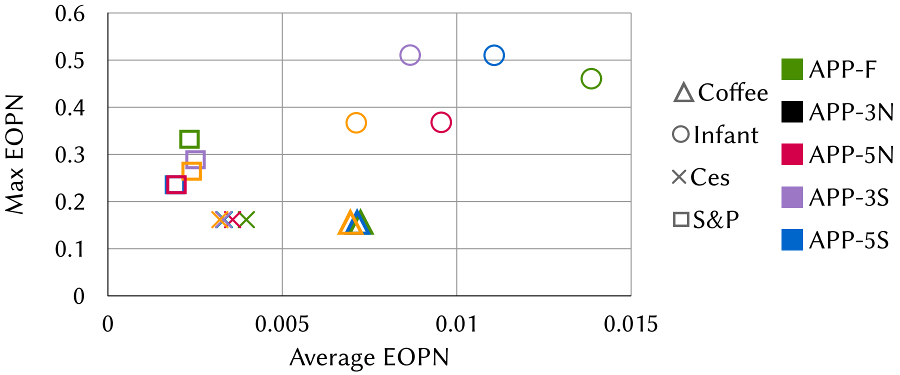


**Figure 3.10** Scatterplots of mean and maximum normalized excess overlap (EOPN) for each APP variant and dataset combination.

We conclude that our mask-aware algorithms do not negatively affect aspect ratio and hence also maintain a high quality for visualizing $\mu$. Thus, we may judge the quality of our algorithms based on mask quality, as performance in aspect ratio is similar.

**Mask quality** We now consider the effect on mask quality as measured by excess overlap. Figure 3.10 shows the mean and maximum EOPN measure for each APP variant. We observe that the datasets cluster clearly within the chart indicating a strong effect of the dataset itself. We may attribute this to the measure not being normalized, yet such normalization is difficult to achieve reasonably: the actual maximum achievable depends not only on the number of nodes, but also on hierarchy depth and structure as well as the (relative and absolute) uncertainty values.

Not surprisingly, the APP-*N variants generally outperform the APP-*S variants, as we measure normalized excess overlap and also use this to heuristically guide the layout algorithm. APP-3N performs best on all datasets except for S&P on which APP-5S performs best, very closely followed by APP-5N. The differences are most clearly observed for the Infant and S&P datasets, but also marginally present in the others.

Let us now briefly consider the effect of our parameter $q$. Conceptually, increasing its value gives us more slack and thus room to improve mask quality. However, we observe that this parameter does not quite achieve that effect in practice: APP-3* versions quite consistently outperform APP-5* versions in mask quality (again, for

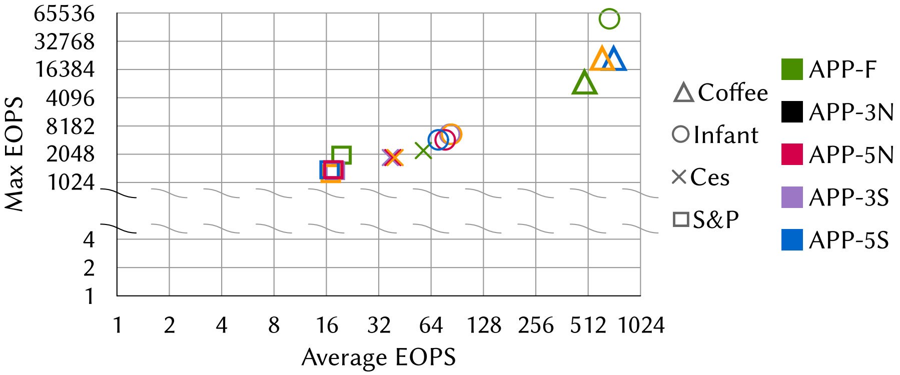


**Figure 3.11** Scatterplots of mean and maximum normalized excess overlap (EOPS) for each APP variant and dataset combination.

the S&P dataset). This can likely be attributed to the extra slack admitting a different split high in the hierarchy, when the estimate is likely further off from the excess overlap that is eventually achieved.

**Area-based quality** We now briefly consider EOPS as it is not strongly correlated to EOPN. Figure 3.11 shows the mean and maximum EOPS for each APP variant. Our first observation is that for the Coffee dataset, APP-F performs best. This seems to be caused by an unfortunate split in the mask-aware algorithms at the root level. In contrast, for the other datasets APP-F performs considerably worse by a factor 17 for Infant and factor of around 1.17 for CES and 1.34 for S&P. Surprisingly, APP-*S is quite consistently outperformed by APP-*N even though the latter uses an estimate not matching the measure we consider here. This is due to the more severe changes made at the higher levels by APP-*S which turn out later to be detrimental to the overall quality.

**Conclusion** We recommend APP-3N as the best algorithm to compute Uncertainty Treemaps: it generally provides the best or otherwise competitive performance in terms of aspect ratio and mask quality (both normalized and area-based) and demonstrates more consistent performance between datasets.

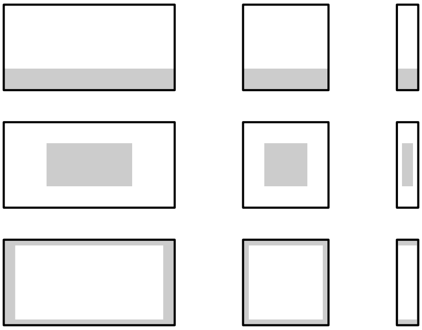


**Figure 3.12** Options for mask placement: (top) bottom rectangle, (middle) inner rectangle, and (bottom) passe-partout. Each row shows nodes with different aspect ratios and 25% relative uncertainty.

## 3.5 Discussion

We now discuss our visualization approach and potential alternatives. In particular, we consider alternatives for mask shape, placement, and rendering which easily combine with our layout algorithms. We also discuss effects of high uncertainty and unbalanced hierarchies.

### 3.5.1 Mask shape and placement

A mask is a generic representation of uncertainty using the area covered by it. This still leaves a big design space for precise shape and placement. We recommend placing rectangular masks at the bottom and along the full width of a node. All examples so far used this approach, as it simplifies relative uncertainty assessment. However, other variants are possible, for example, placing the mask at the center, or as four rectangles along the boundary. We denote the latter type as *passe-partout*, because it resembles this matting construction from picture framing. Figure 3.12 illustrates our recommended bottom rectangle placement and the two alternatives.

The three placement variants have different characteristics. The inner-rectangle placement has the advantage that it is symmetric within the node. However, it has several shortcomings. First, because the inner part is not visually connected to the node we expect that it is harder to relate the mask to its containing rectangle if many nodes overlap. Second, the visual encoding of uncertainty now fully relies on the accuracy of area perception. Unfortunately, the human visual system is not ideal for area estimation. Steven's original power law already indicated a nonlinear mapping between physical area in the stimulus and perceived sensation [127, 128]. Later work discusses other aspects of area perception [76, 134]. In essence, area encoding is not the best option for a reliable visual representation.

Length perception is more accurate following Cleveland and McGill [30] or Mackinlay [80]. With bottom-rectangle placement, a reader can use length perception to estimate relative uncertainty. Comparisons between nodes with identical height (horizontally adjacent) can use height directly to accurately compare relative uncertainty; similarly, height can be used directly to compare absolute uncertainty between nodes with identical width (vertically adjacent). Inner-rectangle and passe-partout placements cannot directly rely on length perception for estimating relative uncertainty. Deciding which node has higher relative uncertainty is possible using length perception if the width or height of the nodes is the same. But the lengths to be compared are more separated (especially for inner-rectangle) compared to bottom-rectangle placement.

Finally, aspect ratios of the masked areas tend to already be low in bottom-rectangle placement. This is exacerbated in the passe-partout placement. As extreme aspect ratios hinder comparing area [70], this is undesirable.

Therefore, for robust uncertainty visualization that supports relative as well as absolute uncertainty assessment we recommend bottom-rectangle placement. However, the passe-partout placement might be preferable if a more symmetric visualization is required, for example, to compare along the horizontal and vertical axes of the treemap, or to implicitly emphasize the nesting hierarchy by the uncertainty borders. Figure 3.13 shows a passe-partout rendering of the Coffee dataset, to be compared with Figure 3.1.

Completely different shapes might also be used such as piece-wise linear or even curved mask boundaries. The mask might also be split into disjoint regions. With this flexibility other optimization cost functions could be explored. However, such shapes differ significantly from the general look of rectangular treemaps and we thus expect that this does not result in a harmonious whole.

### ▸ 3.5.2 Mask rendering

Besides mask layout, its rendering is critical for an appropriate visualization and should be matched to hierarchy level for visibility. We chose slanted hatching; below, we discuss two other options.

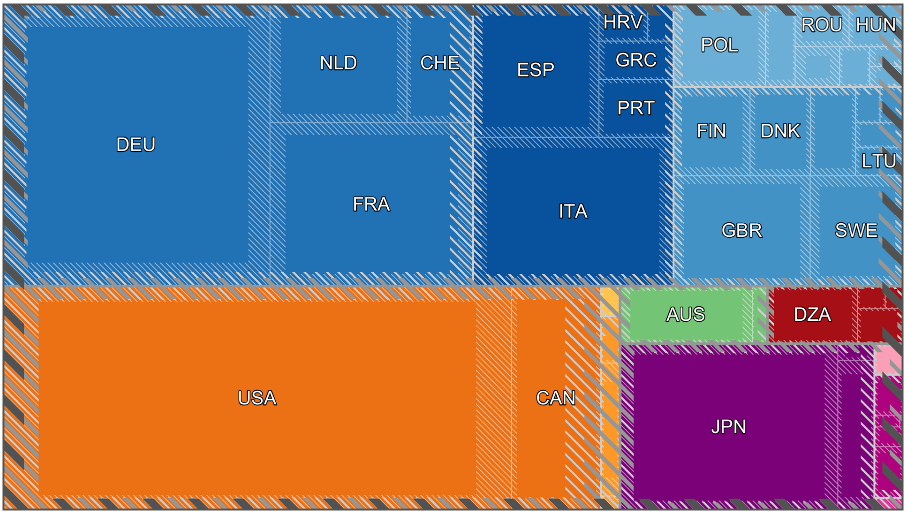


**Figure 3.13** Coffee dataset using APP layout with passe-partout mask.

**Bar hatching** We could use vertical bars in an otherwise identical design. However, such vertical bars are less visually separable from the treemap partitioning itself due to the use of the same angles. Also, we expect bars to give the impression that the uncertainty value is encoded as height instead of area.

**Checkerboard** This design uses a checkerboard pattern of opaque and (fully) transparent squares. The size of the squares doubles with every level of the hierarchy ensuring that the next-level mask can fit a 2 × 2 pattern with a transparent square. Thus, lower-level masks always remain partially visible. The drawback of this design is that it occludes a significant portion of the lower-level masks, as it is also aligned with the treemap partitioning. Moreover, the visual complexity of the pattern is high, possibly distracting from the actual data.

All three variants follow general design recommendations for uncertainty visualization: they implicitly modify the perceived luminance and saturation in the mask areas; these two visual channels are suitable to show uncertainty as discussed by Guo *et al.* [53] or MacEachren *et al.* [79] for example. Our techniques also resemble screen-door transparency for uncertainty visualization [108]. Sketchiness would have been a different kind of visual mapping, but as Boukhelifa *et al.* [19] point out, might lead to an unprofessional look.

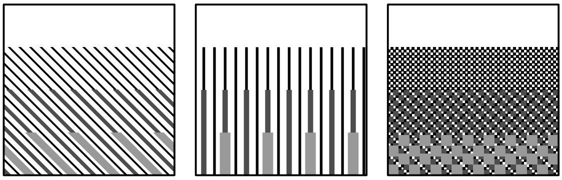


**Figure 3.14** Mask rendering: hatched (left), bar (middle), and checkerboards (right). All show the same three-level hierarchy in a single node.

Figure 3.14 compares our choice and the two alternatives. It suggests that (diagonal) hatching yields a good mask rendering, especially for several layers of overlay. Future work may include a formal evaluation of these and other rendering options in terms of effectiveness for conveying uncertainty in treemaps.

### 3.5.3 Uncertainty and hierarchy

**Uncertainty models** We generally assume that uncertainty is defined at the leaves as a standard deviation and that the underlying distributions are independent. We use this in establishing our datasets, but our techniques (masks, mask quality, algorithms) are otherwise agnostic to the source and relations between uncertainty. Of course, using different models of uncertainty may influence the performance of the various improvements.

For example, each node may be characterized by an interval of potential values. We could use the upper bound as $\mu$ and the interval size as $\sigma$. The aggregation of values through the hierarchy has to be done using interval arithmetic. Though the aggregation method changes, no changes to the techniques are necessary.

**High uncertainty** Our method is designed for scenarios where $\sigma(v) \leq \mu(v)$ for any node $v$. As treemaps are used for non-negative data, this seems a reasonable condition but may not always be met for all nodes. In rare cases, we observe that the uncertainty is larger than the data value. In such cases, the mask covers the full node, though that does not quite accurately capture the actual uncertainty. It does strongly signal nodes with (very) high uncertainty that warrant further investigation, and as such may be sufficient.

To show such high uncertainty more precisely, we may re-mask the node. That is, we show the hatched mask over the entire node, and then appropriate apply a second mask with area $\sigma(v) - \mu(v)$ using a hatched pattern using diagonals in the other

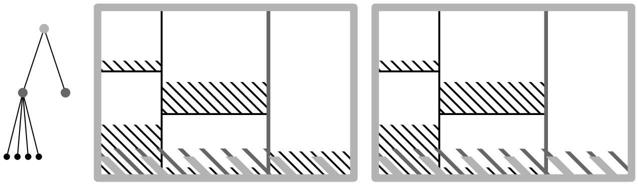


**Figure 3.15** Two different masking strategies for an unbalanced hierarchy (left): height-based (middle) and depth-based (right). Note the different rendering of the mask in the rightmost leaf.

direction. This effectively creates a cross-hatching where the cross-hatched area of the mask counts twice toward visualizing the uncertainty. This allows visualizing uncertainty up to twice the data value of a node.

Generally, we could allow $k$ orientations of hatching to communicate uncertainty up to $k$ times the data value. However, such patterns would likely obscure too much of the node itself making it more difficult to assess its data value and compare it to other nodes. Hence, we do not apply such generalized cross-hatching.

**Unbalanced hierarchies** Thus far we have shown only hierarchies in which the leaves have the same depth. All our real-world datasets indeed have this property. However, more generally, some leaves may have more ancestors than others. We can readily apply our algorithms and masks, but it raises one question: which pattern do we choose for the uncertainty mask? Specifically, do we use the *depth* or *height* of a node in the hierarchy? See also Figure 3.15.

Using depth, going down one level in the hierarchy results in the same mask level. Thus, children of the same parent (or nodes on the same level) have the same mask detail. However, leaves do not necessarily use the most fine-grained mask, possibly de-emphasizing them as leaves. Using height, each leaf has the most fine-grained mask. However, a node with children of different height then uses a mask detail that may be coarser by multiple steps compared to some of its children. The benefits and drawbacks of these variants warrant further consideration in future research. We expect that height-based masks are more efficacious, as they mark leaves more clearly: starting at a coarser mask may incorrectly suggest that the mask is part of a higher-level node and that the leaf has no uncertainty.

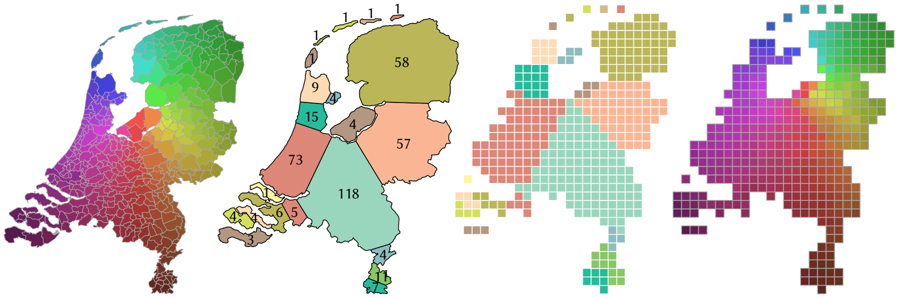
## 3.6 Conclusion and future work

We introduced a technique for the coherent visualization of uncertain hierarchical data. To ensure coherence, we visualized both the uncertainty and the data value using area while maintaining the strict recursive partitioning nature of treemaps. To do so, we designed hierarchical uncertainty masks and applied them to rectangular treemaps. A brief expert review suggests that this technique is practically usable. We modified existing rectangular treemapping algorithms, and experimentally showed that these algorithms achieve high quality in terms of both aspect ratio and mask quality.

We also discussed various alternatives that uncover potential challenges and directions for future work. This includes different mask designs, case studies, and user studies. Another venue for future research is a level-of-detail rendering to make Uncertainty Treemaps scalable for very large datasets—in terms of number of data points (nodes) and the depth of the hierarchy—for example through data-driven pruning of the hierarchy and adaptive mask rendering. Our technique, with its general concepts and metrics, provides a framework upon which such future research can be built.

# Chapter 4

# A Simple Pipeline for Grid Maps



Input Decomposition Mosaic Cartogram Grid Map

**Figure 4.1** A 3-step pipeline to automatically compute a grid map, illustrated on the Dutch municipalities: (1) decompose the containing shape into parts; (2) compute a tile-based Mosaic Cartogram using these parts; (3) assign elements to tiles per part.

A grid map is an established and effective spatial schematization technique. Each spatial element in a grid map, such as a region or a site, is schematized into the same simple *tile* – often a square, hexagon, or other geometry that easily tiles the Euclidean plane. These tiles are then arranged in such a way as to reflect important characteristics of the spatial dimension often using whitespace to capture salient local features. Grid maps are used to visualize geospatial data by popular news outlets [11, 16, 26, 34, 92, 95, 96, 138], discussed by mapping enthusiasts and professionals [43, 100–105, 110, 154], and applied in the academic literature [52, 119, 122, 155, 157].

The tiles of a grid map allow for visualizing data in a small-multiples style as popularized by Tufte [140]. Small-multiples are data-dense visualizations that juxtapose frames in a grid structure. Each frame displays the same visualization for different subsets of the data, for example, according to time steps or based on values in a categorical dimension. In a grid map, the spatial dimension determines both the subsets of the data as well as the arrangement of the juxtaposed frames. Guo *et al.* [52] present an intermediate variant, where space is linearized and used as one axis of a traditional small-multiples grid arrangement.

While small-multiples displays typically fill the available graphical space and arrange the frames according to their inherent order (time, data values, etc.), the spatial case presented by grid maps is inherently different. Viewers typically have a mental map of the geographic area and thus aspects such as recognizability and the ability to locate spatial elements based on expected location play a role. An effective grid map is *coherent* with the underlying geographic space: the tiles maintain properties such as contiguity, neighborhoods, and identifiability of the corresponding spatial elements while the grid map as a whole maintains the global shape of the input. Of particular importance are salient local features of the global shape which need to be represented by tiles assigned to the appropriate spatial elements.

**Contributions and organization** In Section 4.1 we review related work and discuss the various facets of coherence in grid maps. As with any spatial deformation, perfectly coherent grid maps are generally impossible and one must make a trade-off between the facets. Computing a coherent grid map is hence a challenging multi-criteria optimization problem. However, the state-of-the-art shows that simple cases such as close-to-uniform spatial distributions or global shapes with few characteristic features can be solved well using simple tile selection and assignment techniques (as long as sufficient care is taken to guarantee that connected input stays connected in the grid map). Our major contribution is the observation that any input can be decomposed into simple cases and that coherent solutions for the resulting simple subproblems can be combined into a coherent solution for the whole.

Based on this observation, we introduce a simple fully-automated pipeline to compute coherent grid maps in three steps; see Section 4.2. Each step is a well-studied problem: shape decomposition based on salient features (Section 4.3), tile-based Mosaic Cartograms (Section 4.4), and point-set matching (Section 4.5). Our pipeline is a seamless composition of existing techniques for these problems; we implement it in an open-source prototype [50]. In Section 4.6 we showcase the resulting high-quality grid maps, demonstrate the efficacy of our approach on various complex datasets, and compare it to the state-of-the-art.

## ▸ 4.1 Problem exploration

Computing grid maps has two main components: ways to capture the various facets of coherence and algorithms to compute the actual grid maps. After exploring coherence in Section 4.1.1, we discuss algorithms in Section 4.1.2 by reviewing existing techniques and variations. But first we define the terminology for this chapter.

**Preliminaries** Our input consists of a set of *spatial elements*. These spatial elements can be either *sites* – specific point locations where data was obtained – or *regions* – typically, administrative boundaries such as countries or municipalities. In either case, we assume these to be contained in a *containing shape*. For sites, this is provided explicitly, as the (administrative) geographic shape that contains all sites. For regions, we typically assume that the regions partition the containing shape and thus it can be easily derived from the geometry of the regions. Typically, spatial elements are related through a *topology*, indicating pairs of adjacent spatial elements: elements that are connected or neighbors. For sites, these can be supplied explicitly (e.g., road connectivity between cities) or derived implicitly from their locations. For regions, the topology is implicitly represented through the shared boundaries of the geometry. We emphasize that our pipeline does not require a topology to be specified or derived, and a consideration of how such topologies can be derived is beyond the scope of this chapter; we thus implicitly assume that a topology exists in the remainder of this chapter.

We assume that a *tile* is a simple geometric shape that tiles the Euclidean plane. Typical shapes are squares or hexagons. A *tile arrangement* is a composition of a number of these tiles following the tiling of the plane; a tile arrangement selects a number of tiles from the infinite tiling. A *grid map* is a tile arrangement combined with an *assignment*: a one-to-one mapping between tiles and spatial elements.

### ▸ 4.1.1 Facets of coherence

There are myriad facets to coherence and these can be formalized in different ways; see e.g. the work by Meulemans *et al.* [86] for an extensive exposition. For the purpose of this chapter we categorize facets into *local* facets and *global* facets. The former represent facets of coherence with respect to individual spatial elements and their placement within a grid map; the latter capture facets of coherence with respect to groups of spatial elements and even aspects of the tile arrangement with respect to the containing shape. Based on existing work [39, 83, 86, 154], we identify the following facets (see Figure 4.2).

**Local distance [86]:** the distances between spatial elements should correlate to distances between their tiles. *Displacement* [39, 83, 86] is often used as proxy: the distance between a spatial element and its tile should be low. For example, in an overlay of Europe with a corresponding grid map, the distance between (the centroids of) each country and its representing tile should be small.

**Local adjacency [39, 83, 86, 154]:** spatial elements that are adjacent in the topol-

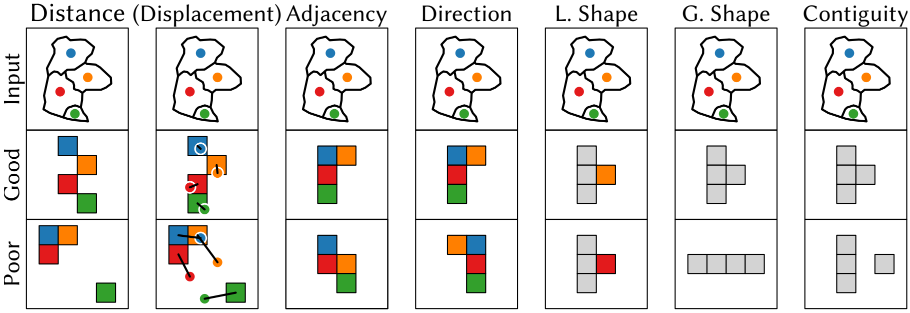


**Figure 4.2** Facets of coherence for grid maps. Displacement is not a facet in itself, but often used as proxy for distance and other facets.

ogy should be assigned to adjacent tiles in the grid map, and vice versa. For example, as Germany borders Denmark but not Italy, the tile for Germany should be adjacent to Denmark's tile but not to Italy's tile in a grid map of Europe.

**Local direction [39, 83, 86, 154]:** compass directions between spatial elements should match the directions between their tiles. For example, as the United Kingdom is north of Spain, the UK's tile should be above Spain's tile. Note that this is a local facet as it considers individual tiles even though the spatial elements and their tiles need not be close in terms of distance. The global position [83] (referred to as location in [39]) of an element in the map can be seen as a variant of local direction.

**Local shape:** tiles on identifiable positions in the tile arrangement should be assigned to spatial elements that belong to that identifiable part in the containing shape. For example, Portugal should be the left-bottom most tile in a grid map of Europe; Florida should be the right-bottom most tile in a grid map of the US states. This facet is not explicitly mentioned in current work, but could be seen as a new interpretation combining the concepts of global position [39, 83] with (global) shape. It is somewhat implicit in the "locate" task [39], though here the consideration of (nearly) full matrices avoids the need for shape in this facet.

**Global distance, adjacency & direction:** the containing shape may have different identifiable parts due to recognizable features or a known subdivision of the containing shape. For example, Germany may be subdivided into its *Bun-*

*desländer* (states) for a grid map of its states. The principles of local distance, adjacency, and direction can be applied to these global parts. As generalizations of local variants, these are typically not explicitly mentioned in literature.

**Global shape [83, 86, 154]:** the overall appearance of the tile arrangement should resemble the containing shape. For example, a grid map of the Dutch municipalities should look like the Netherlands. Note that this facet is agnostic to the assignment of spatial elements to tiles.

**Contiguity:** this global facet captures topology on a coarser level. Specifically, contiguous parts of the containing shape should be represented by a contiguous set of tiles in the arrangement, and all and only spatial elements within the contiguous part should be assigned to this set of tiles. Note that contiguity does not require a topology to be given as it relies on the containing shape. For example, in a grid map of the US states, tiles for Alaska and Hawaii should not touch any other tiles; the tiles for the remaining contiguous states form one contiguous shape in the grid map. This criteria seems to be implicit in the considerations of Meulemans *et al.* [86] when discussing global shape. Various facets can be captured through adequate measures, see [39, 83, 86, 154]. However, global and local shape are particularly difficult to capture while these facets are crucial to solving complex cases; see also our discussion Section 4.6.

Observe that satisfying all of the above facets is generally impossible. For example, not all adjacencies can be represented, distances will need to distort, and there may simply not be enough tiles to show all characteristic features well. Moreover, some facets readily conflict. For example, contiguity generally conflicts with direction and adjacency, adjacency in turn can conflict with direction, and shape (either local of global) can conflict with distance. Generally, we aim to optimize a trade-off between these facets. We consider contiguity to be a hard constraint, one that must be satisfied, as visual discontiguities are salient features useful to identify e.g. islands. Unnecessary discontiguity thus has a high negative impact on local shape.

### ▸ 4.1.2 Algorithms

The algorithmic problems arising from computing grid maps have been studied in various forms. Many specific formulations are computationally hard: for example, optimizing for even a simple version of local topology is NP-hard [21], and optimizing global shape under contiguity constraints is NP-hard as well, even to approximate [20, 75]. Minimizing local directions with a given grid of tiles can be approximated [39], but its computational complexity is open.

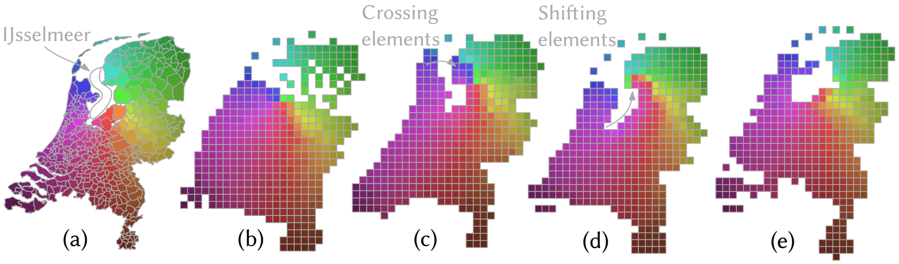


**Figure 4.3** (a) Dutch municipalities. (b) Tile arrangement which minimizes displacement is discontiguous. (c) Tile arrangement with good global shape, followed by displacement minimization. Note the municipalities crossing the characteristic gap in the center (the IJsselmeer) from west to east. (d) Tile arrangement optimized for displacement under the geodesic distance. Now the municipalities from the west shift around the IJsselmeer. (e) Our pipeline: mild deformation in the west to accommodate its municipalities while maintaining the characteristic global shape.

**Simple cases** If the tile arrangement for a grid map is a complete grid – the classic small-multiples setting – and the input has a close-to-uniform spatial distribution (for example, the majority of the *départements* of France have roughly the same area), then computing a grid map readily reduces to a one-to-one point-set matching (of the region centroids to the grid) [39]. Given an alignment which projects the input map onto the grid map, minimizing the displacement of tiles measured by the sum of Manhattan ($L_1$) or Euclidean distances can be solved efficiently [146]. Computing an optimal alignment in terms of translation or scaling is possible, though at significant computational cost [39]. The sum of squared Euclidean distances ($L_2^2$) can be minimized efficiently as well including a simple optimal alignment in terms of translation [31]. The experiments by Eppstein *et al.* [39] show that optimizing $L_2^2$ also results in good local directions and adjacencies for a complete grid. This approach works well for other simple cases, for example, roughly convex containing shapes, such as the boroughs of London. These findings are corroborated by Meulemans *et al.* [86]: if the density of the regions is uniform, or the containing shape is mostly convex, or a mostly convex set of tiles is preselected, then aligning and minimizing displacement under $L_2^2$ yields coherent grid maps.

**Complex cases** The situation is significantly more difficult when the containing shape has salient characteristic features, such as the IJsselmeer in the Netherlands (see Figure 4.3(a) – the large body of water in the middle), the Florida pan handle,

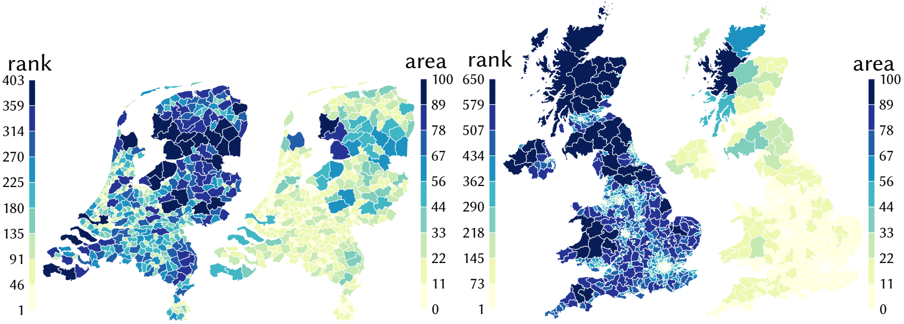


**Figure 4.4** Strongly varying region sizes in the Dutch municipalities (left) and UK constituencies (right). Regions are colored according to nine bins of equal quantiles (rank,left) or of equal area, indicated as a percentage of the largest region (area,right).

or the west coast of the UK. A coherent grid map for such cases should use a tile arrangement that captures the global shape well. A priori, it is not clear how such a tile arrangement should be chosen from the infinite tiling. One could consider selecting the tiles that result in the assignment with the least displacement. Unfortunately, this approach generally leads to discontiguous tile arrangements: see Figure 4.3(b). Meulemans *et al.* [86] show how to compute tile arrangements which make good use of whitespace (unselected tiles) and exhibit good global shape. However, if the region distribution is not uniform, then the subsequent tile assignment minimizing displacement under $L_2^2$ results in severe violations of local shape, see Figure 4.3(c): municipalities from the dense west of the Netherlands travel across the IJsselmeer (municipality sizes as a proxy for density are shown in Figure 4.4(left). Using a more topology aware distance measure such as the geodesic distance does not alleviate the problem: see Figure 4.3(d): municipalities now shift around the IJsselmeer.

The major challenge to resolve is the combination of varying density of spatial elements with characteristic features of the containing shape. Note that this combination is certainly not unique to the Netherlands and is also present e.g. in the UK constituencies: see Figure 4.4(right) for a density overview.

**McNeill and Hale [83]** To the best of our knowledge, McNeill and Hale [83] present the only automated method that aims to explicitly address the problem of varying densities of spatial elements. Their method translates the spatial elements (regions only) and their topology into a graph. To obtain uniform distances between

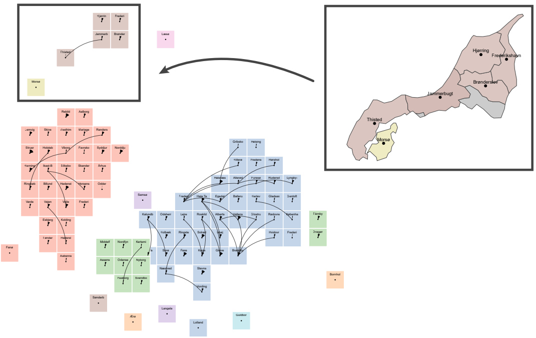


**Figure 4.5** Output of [83]: the municipalities of Denmark, the 5 brown contiguous regions in the north are mapped to discontiguous tiles.[1]

neighbors they employ an approach that uses a form of spring-embedder: vertices are attached by springs to both their neighbors in the graph and to the centroids of their corresponding regions. The containing shape is deformed along with the vertices; the authors find a scale factor such that the deformed shape contains exactly the right number of tiles. Finally, the regions are assigned to tiles using a point-set matching algorithm to minimizes displacement under $L_2^2$.

There are two major drawbacks of their method. (1) The tile arrangement is not necessarily contiguous, even if the input is contiguous. When scaling the deformed shape, the authors ensure only that it contains the correct number of tiles, but not that these tiles are actually connected. See, for example, the five brown regions in Figure 4.5 (from an online implementation[1] by McNeill). (2) Characteristic features of the containing shape are eroded if parts need to grow or shrink considerably due to varying density thereby reducing global and local shape. See, for example, Figure 4.6 (Figure 6 from [83]) which shows the local authorities in the UK: Scotland has been eroded and Wales lost most of its characteristic coast line. In comparison, our pipeline (see Figure 4.7) preserves global and local shape.

McNeill and Hale focus on achieving a uniform distribution – a “simple case” – but they do so for the complete containing shape at once and hence lose control (to some degree) over contiguity and shape. In contrast, our pipeline uses decomposition into simple containing shapes to arrive at simple cases which offers us more control over global and local shape and allows us to guarantee contiguity.

---

[1] `https://observablehq.com/@gjmcn/make-a-tile-map`, 28.04.2020

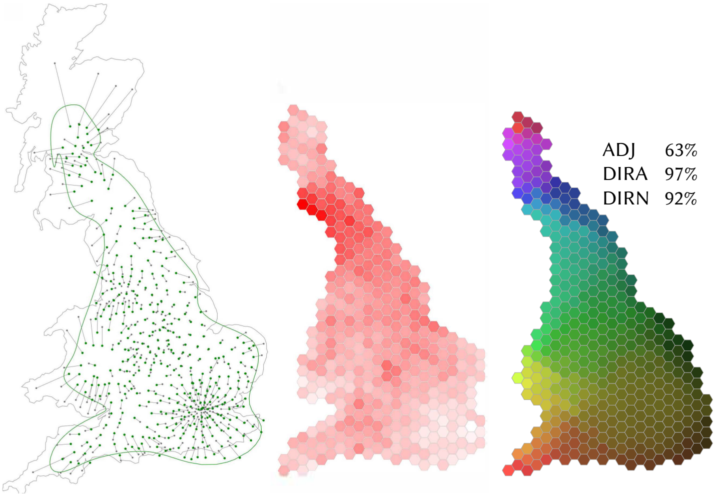


**Figure 4.6** UK local authorities. Left+Middle: Figure 6 from [83]. Right: Reproduced and colored by authority locations. Scotland is eroded.

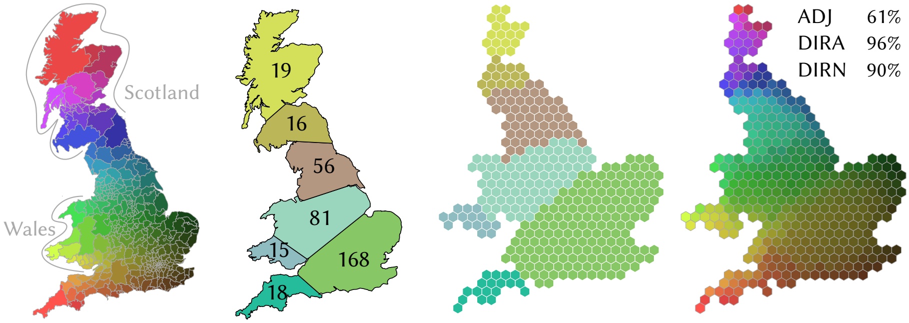


**Figure 4.7** Hexagonal grid map of the UK local authorities using our pipeline: global and local shape are maintained.

**Cartograms** In some sense a grid map is a cartogram where every spatial element (region) has unit weight. There are many cartogram techniques which focus on optimizing e.g. adjacencies, aspect ratio, and cartographic error (difference between the target size of a region and the actual size in the output). However, most cartograms are neither aligned on a grid nor able to produce the desired tile shape. One could of course consider to rasterize the output of the density-equalizing cartogram algorithms proposed by Gastner and Newman [44] and by Gastner *et al.* [45]. However, it will generally be impossible to guarantee contiguity, as smaller regions tend to become rather elongated[2]. Spatially Ordered Treemaps [156] combine ideas of tree maps and cartograms to show hierarchical data; also non-space-filling variants exist [120]. However, such tree maps do not align with tiles on a grid.

One type of cartogram, however, is in principle well suited for the computation of grid maps: the *Mosaic Cartograms* by Cano *et al.* [25] represent regions with multiple tiles where the number of tiles corresponds to (integer) data values associated with each region. As such, Mosaic Cartograms cannot directly be used to compute coherent grid maps, but they are an integral part of our pipeline.

**Semi-automatic** Wongsuphasawat [153] describes a semi-automatic pipeline to create grid maps. The first step computes a coarse, discontiguous grid map using overlap removal and grid snapping. The result has significant discontiguities which are then patched in a second step by manually shifting parts of the arrangement until the result is contiguous and has a suitable global shape.

## 4.2 A 3-step pipeline for coherent grid maps

Our discussion in the previous section points directly towards a natural pipeline to compute coherent grid maps: decompose the problem into one or more simple cases and then apply point-set matching. The simple case our pipeline achieves is arguably the simplest – that of convex containing shapes: we decompose the containing shape into relatively simple, mostly convex parts. Then we compute a Mosaic Cartogram [25] based on the parts and the number of spatial elements in each part. The Mosaic Cartogram grows or shrinks each part accordingly while ensuring contiguity and maintaining shape: the tile arrangement per part is thus mostly convex as well. Point-set matching for the tile arrangement of each part completes the pipeline. Concretely, we use the following three steps, each of which is discussed in detail in the subsequent sections (see also Figure 4.1).

---

[2] See `https//www.worldmapper.org`, accessed April 2020.

**Step 1 – Decompose into parts:** we decompose the containing shape based on salient features into simple, mostly convex parts and note how many and which spatial elements each part contains.

**Step 2 – Arrange tiles:** we compute a Mosaic Cartogram, based on the parts, using the number of spatial elements as weight. The Mosaic Cartogram by design maintains shape, adjacencies, directions, and also ensures contiguity. The result is a tile arrangement where each tile is associated with one part and each part is represented by the correct number of tiles.

**Step 3 – Assign elements:** per part, we use point-set matching to compute an assignment that minimizes the sum of squared Euclidean distances between the spatial elements and the tiles.

Each step of the pipeline is thus a well-studied problem. In principle, different techniques could be used for each step – especially for Step 1 – though the literature suggests that our proposed composition is effective. To seamlessly create the complete pipeline, we made small changes to existing techniques or simplified existing approaches.

## 4.3 Decompose shape based on salient features

The first step of the pipeline decomposes the containing shape into parts based on salient features. We can treat each contiguous part of the input separately, and hence can restrict our attention to the decomposition of a simple polygon $P$. Our output are the parts of $P$ together with the spatial elements contained in each part.

**Related work** Shape decomposition is an extremely well-studied problem. Below we sketch some of the most important considerations and approaches. An important related field is that of object recognition which considers many aspects of objects such as shape, color, texture, motion, context, etc. It has been observed, though, that humans can often recognize an object solely by its shape [60]. This recognition is based on a decomposition into visually salient parts [60, 94, 114, 117, 118], via so-called *cuts*: straight boundary-to-boundary line segments fully within the shape.

Based on psycho-physical findings a number of rules and constraints have been proposed in the literature to mimic the human visual decomposition in finding cuts. Though more general, we present them here for the case of simple polygons, as this is the setting for our pipeline. The most recognized of these rules are the minima rule [60] (a cut should always start at a reflex vertex) and the short-cut rule [118] (shorter cuts are preferred over longer cuts if they are otherwise equivalent).

To determine candidate cuts, the definition of a part-cut [117] is often used which stipulates that a cut should cross a local axis of symmetry, in addition to being contained within the shape and following the minima rule. Local symmetry can be seen as a weak form of symmetry: the two sides along each axis of local symmetry are shape-wise similar, but are not strictly symmetric. The medial axis [17] is often used for this purpose (e.g. [94]), but different formulations are possible.

Typically, there are many candidate cuts, and thus there is a need to filter and select an appropriate set of (final) cuts, see for example[73, 78, 88, 94]. Our implementation is inspired by the work of Papanelopoulos *et al.* [94] who use the medial axis to construct a candidate set, which is hence based on local symmetry.

**Our implementation** Using the medial axis to identify candidate cuts guarantees that all cuts are pairwise disjoint. Hence, any combination of candidate cuts yields a valid decomposition into parts. Let $\mu(P)$ denote the medial axis of our input polygon $P$, and consider a single segment $s$ of $\mu(P)$. Note that $\mu(P)$ consists of both straight line edges and parabolic arcs. Let $g_1(s)$ and $g_2(s)$ denote the two shortest distinct line segments along the boundary of $P$, such that all inscribed circles of $P$ with their center on $s$ touch both $g_1(s)$ and $g_2(s)$. These line segments may have length 0 and thus be a vertex. We add a candidate cut if $g_1(s)$ or $g_2(s)$ is a reflex vertex. If both are reflex vertices, $s$ is a straight line segment and we add a candidate cut between $g_1(s)$ and $g_2(s)$ (see Figure 4.8(a)). If only one is a reflex vertex and the other a line segment, then $s$ is a parabolic arc and we add two candidate cuts from the reflex vertex to the two endpoints of the line segment (see Figure 4.8(b)). Since at least one endpoint of each cut is a reflex vertex, all candidate cuts follow the minima rule.

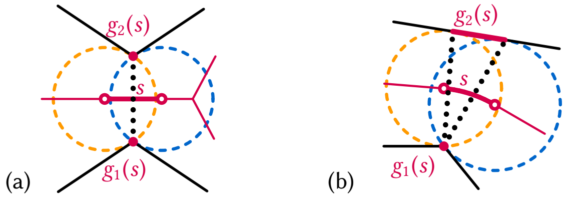


**Figure 4.8** Candidate cuts (dotted lines): medial axis segment $s$ is (a) a straight segment, or (b) a parabolic arc. Inscribed circles (blue and orange) centered on endpoints of $s$ define line segments $g_1(s)$ and $g_2(s)$.

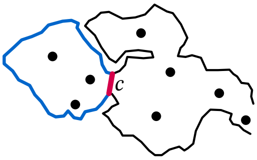


**Figure 4.9** Candidate cut $c$ with dilation 14.5 and productivity 3. Shortest length along the boundary in blue; dots represent spatial elements.

Our filtering scheme uses a dilation threshold $d$ and a productivity threshold $p$ (see Figure 4.9). The dilation (or detour factor) of a cut is the shorter of the two lengths along the boundary of $P$ divided by its Euclidean length. This ratio is always at least 1 due to triangle inequality. The productivity of a cut is a new concept, tailored to the construction of grid maps. Each cut partitions (the centroids of) the spatial elements into two sets. The productivity of a cut is the number of elements in the smaller set. Our pipeline does not allow cuts with low productivity to ensure that each part has sufficient tiles at its disposal to achieve good local and global shape.

We select candidate cuts as follows. First, we sort them by increasing length following the short-cut rule and then process cuts one-by-one. If the dilation of a cut is above $d$ and its productivity is above $p$, then we apply the cut: we partition $P$ and its spatial elements and recurse on the resulting two subpolygons. In this recursive step we update both dilation and productivity of each candidate cut with respect to its new subpolygon. See Figure 4.10(b) for an example of the eventual result.

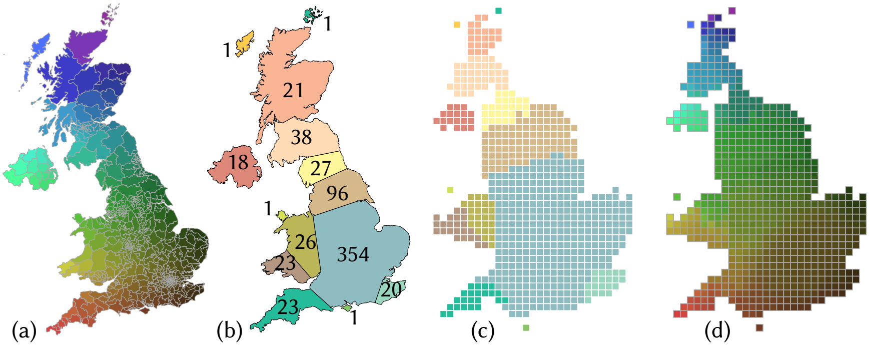


**Figure 4.10** UK constituencies, computed with dilation $d = 3$ and productivity $p = 20$. Numbers in (b) indicate number of spatial elements per part.

Let $n$ denote the complexity of the containing shape and $m$ the number of spatial elements. As the medial axis takes $O(n)$ time to compute and has linear complexity [29] we compute our $O(n)$ candidate cuts and sort them in $O(n \log n)$ time. Per candidate cut we keep track of the length along the (sub)polygon boundary and the number of elements on both sides of the cut, from which we can derive dilation and productivity in $O(1)$ time. Initializing these values takes $O(n^2 + nm)$ time in total using a straightforward implementation. When a candidate cut is selected, we can update these values in $O(1)$ time each. We leverage the medial-axis structure to traverse the candidates on both sides and update all values in $O(n)$ total time. As such, the selection process can be executed in $O(n^2)$ time. The bottleneck is hence initializing the selection process in $O(n^2 + nm)$ time.

**Using other methods** Our specific implementation can easily be replaced by other algorithms that detect cuts based on salient features. However, for the next step in our pipeline it is important to avoid cuts with low or even zero productivity. A part with zero spatial elements cannot be represented by a Mosaic Cartogram and will hence violate topology and thereby contiguity.

## 4.4 Arrange tiles using mosaic cartograms

The second step of our pipeline uses the parts – the decomposition result – to compute a tile arrangement. We do so for all parts simultaneously to ensure that tiles do not overlap, to ensure contiguity, and to optimize other global facets of coherence. The inputs for this step are the parts, that is, a subdivision of the simple polygon $P$, together with the number of spatial elements that each part represents. The output of this step is a Mosaic Cartogram: a contiguous tile arrangement, including a mapping from parts to contiguous sets of tiles of the correct cardinality.

To the best of our knowledge, the original algorithm for Mosaic Cartograms [25] is currently the only method to solve the given problem. However, other algorithms could in principle replace this technique as long as they can guarantee contiguity and represent each part with the exact number of tiles – see also Section 4.6.

**Our implementation** We use Mosaic Cartograms [25] which deform in a shape-aware manner while maintaining contiguity and global adjacencies between the parts, and optimizing for preserving global directions between neighboring parts; see Figure 4.10(c) for an example result.

A Mosaic Cartogram is roughly computed as follows (refer to [25] for details). First, an initial tile arrangement is computed, such that each part has the correct adjacencies without accounting for shape or the necessary number of tiles. Then, for each part, the algorithm computes a guiding shape: a contiguous set of tiles that resembles the shape of the part. The tile arrangement is then iteratively modified by moving guiding shapes and changing tiles while ensuring that adjacencies are maintained. Movement is based on the desired direction between two parts, and tiles are changed to also converge on the required number. This process stops when the guiding shapes no longer move and tiles no longer change. At this point the exact number of tiles per part might not yet be achieved, but typically it is close. A final post-processing step corrects the number of tiles and fills any unwarranted gaps in the tile arrangement while still maintaining the correct adjacencies.

The correct number of tiles and correct adjacencies might not be compatible (e.g., a part with only a single tile that needs to be adjacent to many other parts). Indeed, this can be expected with a low productivity threshold in the first step, though parts with few spatial elements typically have few adjacent parts. Whereas Mosaic Cartograms opt for a slight deviation in the number of tiles (cartographic error) to ensure the right adjacencies between parts, this is not an option in our pipeline: each part must be represented by as many tiles as the number of spatial element it contains.

If every part is adjacent to the "outside" (i.e., the adjacency graph is outerplanar), the post-processing step has significant freedom, and typically sufficiently so to achieve the right number of tiles. This is the case in all our test instances; without subdivisions (see Section 4.6) the topology is even a tree by design of our decomposition step. We therefore did not observe any deviation from the exact number of tiles, even while perfectly representing adjacencies of the parts.

If needed, one could add a second post-processing step similar to the first; but rather than requiring that adjacencies are preserved, we enforce only that the result remains contiguous. In this way we can ensure that the resulting Mosaic Cartogram has zero cartographic error and is contiguous at the possible cost of small topological violations. It is an interesting direction for future work to compute good Mosaic Cartograms with zero cartographic error more directly.

The algorithm for Mosaic Cartograms assumes a contiguous input. We generate such a cartogram for each contiguous piece of our input and compose them. Our implementation features some automation based on direction. Specifically, we start with the largest piece (for example, mainland UK or mainland NL) and simply use its result. For any subsequent piece (usually islands of decreasing size), we use the direction between the closest spatial element of the new piece towards any of the already-placed pieces. This gives a starting tile to place the new piece from which we search locally for a placement that does not cause overlap or new adjacencies. This automation works reasonably in simple situations (e.g., UK: mainland with six well distributed islands) but requires manual post-processing for maps with more intricate layout (e.g., Netherlands: mainland with seven clustered islands). We apply this optional manual step in all our results, and leave better placement of separate pieces in Mosaic Cartograms to future work.

## 4.5 Assign elements to tiles via point-set matching

All that remains now, is to assign spatial elements to tiles in the tile arrangement computed in Step 2. As the sets of tiles and the number of spatial elements per part match exactly, we can execute this step separately for each part. The input is thus a set of contiguous tiles for one (mostly convex, and hence simple) part together with the spatial elements for that part. The output is the tile assignment for this subset of the spatial elements. Combining all assignments yields the grid map.

**Related work** There are various ways to assign spatial elements to a tile arrangement. For tile arrangements (close to) a grid one can use a greedy assignment as proposed by Wood and Dykes [156] for Spatially Ordered Treemaps and adapted by Eppstein *et al.* [39] for grid maps (called *SpatialGrid* in [39]). Eppstein *et al.* [39] propose a variety of other methods for this case including displacement optimization under the sum of squared Euclidean distances ($L_2^2$). They show that optimizing $L_2^2$ also results in good local directions and adjacencies, and these findings are corroborated by Meulemans *et al.* [86]. Since we can minimize $L_2^2$ efficiently [31] this approach is generally the method of choice and also the one we employ in our pipeline. Later work by Liu *et al.* [74] proposes to use a form of constrained multi-dimensional scaling to assign regions to a grid. However, in addition to the restriction to the grid, the assignment quality is generally lower than the one via $L_2^2$.

**Our implementation** As stated above, we optimize for $L_2^2$ which yields good results in local facets of coherence for the simple cases that the earlier steps in the pipeline create. To align the spatial elements and the set of tiles of a part, we compute the affine transformation such that the bounding box of the set of tiles is equal to that of the centroids of the spatial elements. Subsequently, we use point-set matching from centroid to centroid to minimize the sum of squared Euclidean distances in this transformed space implemented as a linear program [31].

## 4.6 Results and discussion

In this section we investigate the efficacy of our pipeline in three aspects: (1) parametrization of our shape-decomposition step; (2) a comparison to state-of-the art; (3) using a known subdivision of the containing shape. We end this section with a general discussion and future work.

**Parametrization** Contrasting the other two steps in our pipeline, the first step of decomposing the containing shape is parametrized. Setting suitable thresholds is important to achieve the best results. Here we briefly investigate the effect of these parameters via the result of the subsequent step: the tile arrangement.

Figure 4.11 shows results for the Dutch municipalities using a a dilation threshold $d \in \{2, 3, 5\}$ and productivity $p \in \{4, 10, 30\}$. The effect of $d$ is quite straightforward: a lower value of $d$ means that more candidate cuts can be selected and thus results in more identified parts. Productivity $p$ has a similar effect: parts are required to contain more spatial elements and thus higher values lead to fewer parts. With too many parts the computed Mosaic Cartogram is too detailed resulting in a jagged, uneven appearance in places where this is not necessary for global shape – this is because Mosaic Cartograms consider shape purely on a part-level. Yet, with too few parts we do not obtain the simple case within each part due to uneven distributions of the spatial elements while still having recognizable features. Hence, both need to work together to obtain parts that represent simple cases and that are both visually salient as well has having enough elements to be represented reasonably. Of these provided figures, we deem that the case of ($d = 3$,$p = 4$) strikes this balance well and nicely places southern Flevoland (four municipalities in the central brown region in the figure) along the IJsselmeer. We use these settings and hence this Mosaic Cartogram also in Figures 4.1, 4.3(e) and 4.13(c) showing the Dutch municipalities.

Productivity $p$ can likely be configured similarly across instances of similar size, though effects of geography can be expected as culturally significant features bias towards certain decompositions. However, dilation threshold $d$ faces a difficulty arising from a well-known geographic phenomenon: the coastline paradox [82]. In our situation, the detail at which the containing shape is represented greatly affects the length along the polygon boundary, used to determine dilation; yet, the length of a candidate cut is largely unaffected. As a result, the dilation threshold for a particular instance depends on the resolution of the input map. A normalization of the shape boundary is not quite enough, as shape complexity remains a factor and borders may locally vary in semantics and thus in their effect on the local boundary

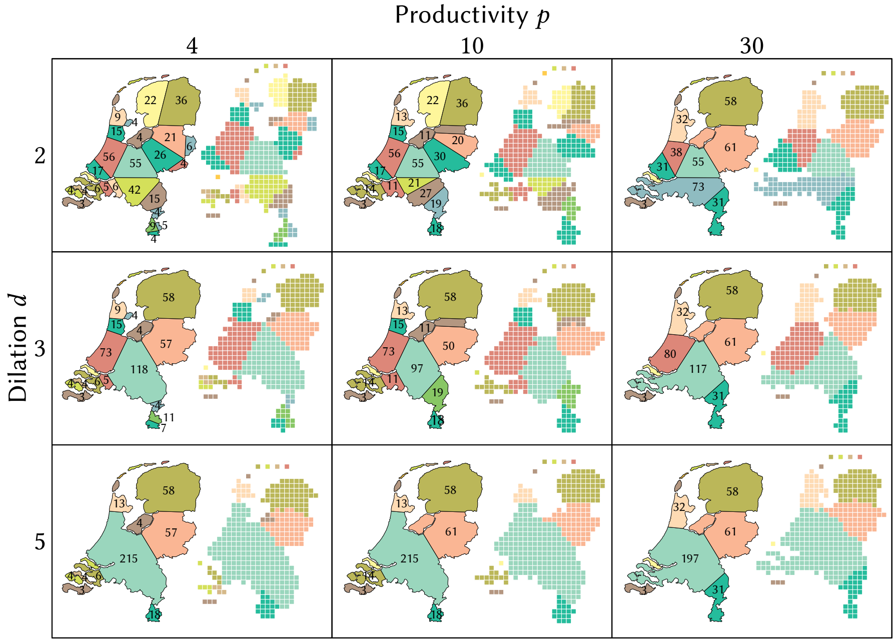


**Figure 4.11** Results of varying productivity and dilation thresholds on the decomposition and subsequent Mosaic Cartogram.

length – consider, e.g., differences between actual coastlines and borders with other countries such as Norway with its fjords and straight border with Sweden. A suitably normalizing simplification of the containing shape might provide a solution to this resolution-dependence; we leave this investigation to future work.

**Comparison** With Figures 4.6 and 4.7 shown earlier, we compare our method to the approach of McNeill and Hale [83], using a hexagonal grid map for the UK local authority districts. These figures clearly show that our pipeline retains more of the characteristic features, especially around Scotland and Wales. The smooth gradient of colors in our result also shows that local facets are preserved well. To further compare the two methods on the local facets of coherence, we use the measures proposed by Meulemans *et al.* [86] for local adjacency and local direction: The percentage of adjacencies maintained (ADJ), the percentage of orthogonal directions maintained (DIRA), and the percentage of orthogonal directions between neighbours maintained (DIRN). The two methods have comparable scores for the UK map. We also provide a grid map with square tiles in Figure 4.10 on the UK constituencies showing that this improved shape is also achieved with other tile shapes.

Figure 4.12(d,h) shows the result of our pipeline on the contiguous states of the US without and with Washington DC, a common example and use case of grid maps. We observe that this case is not necessarily complex as evidenced by the few parts in the decomposition (Figure 4.12(b,f)), but not quite simple either: e.g., Florida and the Great Lakes give characteristic features for local shape that can be preserved, and states along the (north)eastern coast are significantly smaller than the other states. That only few parts are identified signals that our pipeline can adequately cope with simpler cases as well without the earlier steps in the pipeline needlessly distorting the result. That is, it can be applied nearly agnostic to the input beyond configuring the productivity and dilation thresholds as discussed above. We compare our result shown in Figure 4.12(h) with a result of McNeill and Hale [83] shown in Figure 4.12(e) using our color scheme. The maps are roughly of comparable quality which the metrics again further support, but our map has some points that we consider to work specifically well in comparison: Florida is the bottom-rightmost state; Michigan can be seen to be located between the Great Lakes. McNeill and Hale's result on the other hand nicely preserves Washington as the northwestern-most state, and is slightly more compact for the northeastern states.

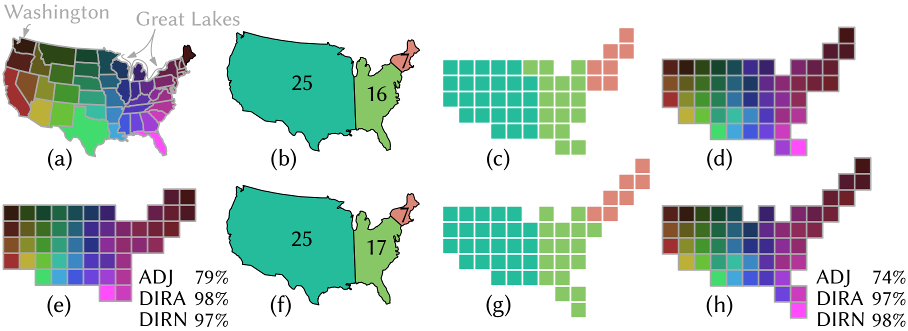


**Figure 4.12** (a) Contiguous states of the US. (b–d) Result of our pipeline ($d = 5$, $p = 4$). (e) Grid map based on Figure 5a in [83], which includes Washington DC. (f–h) Result of our pipeline ($d = 5$, $p = 4$), for input including Washington DC.

Comparing Figure 4.12(d,h), the addition of the single tile for Washington DC has quite an effect on the Mosaic Cartogram and thus the eventual result. Though the overall structure is the same, the shape around the Great Lakes is less clear in (d), but the northeastern states are represented more compactly. It is also worth noting that the projection of the input (here, Albers projection) for such large areas may have an effect on the coherence of the grid map [86].

Our method is somewhat slower to compute. For the 374 UK local authorities, McNeill and Hale report a running time of four seconds [83] for the result in Figure 4.6. Our result shown in Figure 4.7 takes 84 seconds to compute, with 21 seconds spent on generating the partition, and 60 seconds spent on computing a Mosaic Cartogram on a standard laptop. Improving this running time substantially thus requires a faster algorithm to compute Mosaic Cartograms.

**Subdivisions** So far, we have shown inputs of a single containing shape with spatial elements, that is, the containing shape is a collection of disjoint simple polygons. However, the containing shape can often be subdivided into several administrative units: e.g., a country is often subdivided into provinces or states. The question is whether such known subdivisions of the containing shape can be used instead of, or in addition to, decomposition in our pipeline. To explore this, we consider again the Dutch municipalities. As these are hierarchically organized into provinces, we use province boundaries to subdivide the containing shape; see Figure 4.13(a).

Figure 4.13(b) shows the result of applying our pipeline without applying Step 1 – shape decomposition. Instead, we simply use the provinces as parts and start from Step 2. To compare, Figure 4.13(c) shows our pipeline solution without subdividing by provinces, but with the province borders indicated. The main advantages of using the provinces as parts is that (contiguous parts of) provinces remain contiguous in the result, and that adjacencies between provinces are exactly the same to those in the input map as achieved by the Mosaic Cartogram. Without using the province subdivision, this is not guaranteed, and indeed we can observe slight discontiguities in Figure 4.13(c) for some provinces. Note, however, that the provinces are only discontiguous when considering rook's adjacency (sharing a side of the square), but are in fact contiguous in queen's adjacency (sharing a corner is sufficient to be considered connected). As such, the deviation is indeed only minor.

At a glance, using the subdivision improves global shape in some places, such as southern Limburg (the southernmost province). However, this does not readily imply a good grid map: without decomposition, we see the same problems as discussed in Section 4.1.2 now appearing at a smaller scale: the tile indicated by the red arrow in Figure 4.13(b) is assigned to the municipality in Figure 4.13(a) indicated by the arrow

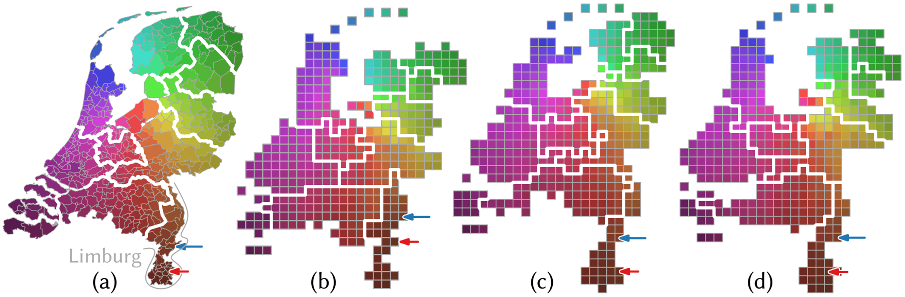


**Figure 4.13** Effect of applying a subdivision, illustrated with the Dutch municipalities and provinces. Thick white boundaries separate provinces. (a) Input map. (b) Result of our pipeline without decomposition, using provinces as parts. (c) Result of our pipeline ($d = 3, p = 4$) without using provinces. (d) Result of our pipeline ($d = 3, p = 10$) by applying decomposition on each province.

of the same color; based on shape one would rather expect this tile to represent the municipality indicated by the blue arrow. That is, this map has poor coherence in terms of local shape. With decomposition, this improves significantly as illustrated by the colored arrows for these same municipalities.

This raises the question whether we can effectively combine these two ideas. That is, rather than applying the decomposition step to the entire containing shape, we refine the subdivision into parts by applying decomposition to each polygon it defines, i.e., to each province in our example. We then obtain parts that respect the subdivision; the Mosaic Cartogram ensures the correct topology between parts and thus we achieve contiguity for each province. The result of this is shown in Figure 4.13(d). Whereas it indeed maintains contiguity of each province (and the entire shape), we see global shape deteriorating slightly, again mostly in terms of smoothness along the boundary. We attribute this to the increased number of parts with typically fewer spatial elements: as a result, the algorithm for Mosaic Cartograms is less sensitive to small changes along the boundary in capturing shape.

## 4.7 Conclusions and future work

In this chapter we introduced a simple 3-step pipeline that generates a coherent grid map for a complex dataset by subdividing the problem into simple cases which can be solved independently. Each step is a well-studied problem that preserves different facets of coherence. The pipeline first decomposes the shape into roughly convex parts based on the salient features of the shape using the medial axis. It then assures each part has the correct number of tiles as well as the correct global shape using Mosaic Cartograms. Finally, it assigns each spatial element to a tile for each part independently using point-set matching that results in high local coherence within the simple part. We verified the efficacy of this pipeline on four different datasets, resulting in high quality grid maps for each of them.

Yet, there is room for improvement, as evident from the discussion. One major benefit of this pipelined approach is that improvements within each step can be carried over to improve grid-map computation – though especially for Step 1 it remains to be assessed how improvements in shape decomposition interact with the subsequent steps. We specifically see potential to improve Step 2. Whereas Mosaic Cartograms produce good tile arrangements in general, there are some limitations to the method. Specifically, the outline of the tile arrangement can capture shape well, but also may cause a somewhat jagged boundary where the containing shape did not necessarily feature such.

This jaggedness stems from three places in the algorithm. Firstly, when determining the guiding shape, jagged boundaries can occur as the rasterization of the region is not perfect and does not explicitly take smoothness into account. Secondly, during the moving and growing of the guiding shapes, the guiding shapes do not have to line up with each other anymore which can result in jagged adjacencies between the parts. While generally avoiding these jagged adjacencies may be difficult, we believe that the situation within the pipeline which ensures a simple set of adjacency requirements between parts may allow for algorithmic improvements to alleviate the problem. Thirdly, when correcting the number of tiles in each part, the algorithm does not take the smoothness of the boundaries into account, and thus extra tiles can appear on the boundary. Again, the case within our pipeline is simpler than the general case for cartograms, and we expect that this problem can be addressed through modifying or improving the underlying algorithms. Alternatively, given that only few tiles need to be moved in our observed results to smoothen the boundary, a simple post-processing step could be sufficient in many situations.

Awaiting further algorithmic improvements, our pipeline also makes it straightforward to interact with the intermediate results: one can manually add or remove cuts in Step 1 that do not quite follow readily from the input geometry, but rather from a user's understanding of the local geography – as it is well known that small geometric features on a map may be relevant from a cultural or geographic point of view. Especially the Mosaic Cartogram can be interacted with easily and in a predictable manner. By simply shifting and swapping tiles, one can steer the input for the final step to an even more polished result. With Step 3 being efficiently computed, this can be done interactively with the tile assignment being updated on-the-fly to ensure the best result. We emphasize that all results in this chapter were not changed manually, beyond improving the placement of separate pieces (islands) of the Mosaic Cartogram (see Section 4.4).

We have compared our results to that of McNeill and Hale [83] qualitatively. A general and impartial quantitative evaluation would be useful, either in the form of user studies or computational experiments. The difficulty in achieving the latter is how to capture each facet of coherence well. Though Meulemans *et al.* [86] provide a suite of metrics, global shape poses a particular challenge: common shape similarity measures are not suitable for handling distortion which is a necessity in obtaining a high-quality grid map. Yet, not all distortion is equal; it must be applied effectively and in a structured way as to still have a sense of local scale for salient features, even though the larger structures are distorted. The ideal of "no distortion" in shape is not only unachievable but also undesirable. That is, current measures cannot easily operate within the context of density differences that force the distortion. Possibly, ideas from focus-and-context maps to measure distortion based on a local scale [35, 58] might be useful in designing a locally scale-aware shape-similarity measure. Local shape similarly poses challenges, as it relies on automatically identifying and relating characteristic features in both the grid map and the input map.

# Chapter 5

# Spatial Set Visualization

Hypergraphs are a powerful structure to represent unordered set systems. In general they consists of a number of elements (vertices) and a number of different subsets over these elements (hyperedges). Visualizing hypergraphs allows one to gain insight into the various set relations between the hyperedges. Hypergraph visualizations can be, roughly speaking, divided into two strands: those where the positions can be chosen by the layout algorithm (e.g. [59, 115, 116]), and those where the position of the elements is fixed (e.g. [1, 32, 36, 85]), for example due to a spatial dimension in the data. Some hypergraph visualizations additionally replicate elements (e.g. [2, 59]) to overcome layout complexity. For a more detailed overview and in-depth classification of set visualization methods we refer to the survey by Alsallakh *et al.* [3]. We focus on a visualization using a single representation for each element and fixed positions to visualize spatial set data.

Additionally, we assume that the representation of two sets may not cross at common vertices which is different from the (often implicit) assumption in theoretical research on drawing hypergraphs (e.g. [24, 65]). Such crossings are usually not deemed problematic as most visual encodings rely on the local *nesting* of intersecting polygons (in line with the prototypical Venn and Euler diagrams [8] and similar visual overlays [32, 36, 85]) to identify set memberships. Nested lines connecting vertices such as employed by [36, 85] give a strong and potentially misleading visual cue of containment between hyperedges. Moreover, the absence of nested lines does not imply that one of the hyperedges is not a subset of the other. As such, this may lead to incoherence between the visualization and the relations in the data.

One of the most well-established quality criteria (see e.g. [97, 99]) for graph drawing is planarity. A planar graph is one that can be drawn without edge crossings; such drawings make it easier to understand the graph structure [98]). Generalizing to hypergraphs this concept remains relevant, but depends on how one visualizes a hyperedge. Using nested encodings as used by [36, 85] the problem reduces to finding a planar support. Deciding whether a planar support exists is possible for some simple support classes (see [24] for a discussion), but quickly becomes difficult as it is already NP-hard for 2-outerplanar support graphs [24].

Representations that do not require nesting are edge-based drawings [81] or the equivalent Zykov representation [151] for which notions of planarity follow readily from the standard notion for regular graphs. One of the representation used in Kelp Diagrams [36] also avoids nesting, but its appearance is cluttered and it does not feature in the later extension, KelpFusion [85]. We suggest an alternative representation that uses *disjoint polygons*. Vertices are represented as simple geometric primitives (e.g. a square or circle); hyperedges are represented as connected polygons that overlap only and all its incident vertices; and all such polygons are pairwise disjoint. As illustrated in Figure 5.1 our disjoint-polygons encoding is stronger than the Zykov encoding, as it can visualize some hypergraphs that are not Zykov-planar, whereas any Zykov-planar hypergraph admits a disjoint-polygons representation. In the disjoint-polygon representation we can use vertices to "pass in between" the representations of other hyperedges, though not as flexibly as is allowed for planar supports: the polygons must remain disjoint.

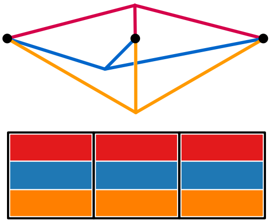


**Figure 5.1** A hypergraph that is not Zykov-planar (top) but has a disjoint-polygons drawing (bottom).

**Contributions** We investigate the properties of drawing hypergraphs using disjoint polygons. Motivated by moving towards a set visualization in a grid map (see Chapter 4), we study the variant where each element has a fixed location, being a cell in a rectangular grid. As an initial exploration we focus on the case with two hyperedges. We identify one hyperedge with the color red, and the other with the color blue. As such, each cell is either red, blue, both (purple), or uncolored (white). We thus aim to partition each purple cell into red and blue pieces such that the resulting pieces of a single color form a connected polygon. In Section 5.1 we define the notation that is used through this chapter. In Section 5.2 we derive a necessary and sufficient condition to efficiently recognize whether an instance is solvable. In Section 5.3 we additionally bound the number of colored pieces within each cell by a small constant for solvable instance, and prove that this bound is tight.

## 5.1 Preliminaries

We define a *k-colored grid* $\Gamma$ as a rectangular grid in which each cell $s$ has a set of associated colors $\chi_s \subseteq \{1, \dots, k\}$. A *fully* $k$-colored grid is the case where $\chi_s \neq \emptyset$ for all cells $s$. Throughout this chapter we primarily investigate 2-colored grids and use *colored grid* to refer to the 2-colored case unless indicated otherwise. We refer to the two colors as $(r)ed$ and $(b)lue$; cells for which $\chi_s = \{r, b\}$ are called $(p)urple$. Cells with no associated colors are *white*.

A region is a maximal set of cells that have the same color assignment ($r$, $b$, or $p$), and every cell $s$ in the region is connected via horizontally or vertically adjacent cells to every other cell $s'$ in the region. A *panel* $\pi_s$ for cell $s$ (with $\chi_s \neq \emptyset$) maps each color $c \in \chi_s$ to a (possibly disconnected) area $\pi_s(c)$ such that these panels partition the cell: that is, $\bigcup_{c\in\chi_s} \pi_s(c) = s$ and $\pi_s(c_1) \cap \pi_s(c_2) = \emptyset$ for colors $c_1 \neq c_2$. A *painting* $\Pi$ of a $k$-colored grid consists of panels $\pi_s$ for each cell $s$ with $\pi_s(c) \neq \emptyset$ for each $c \in \chi_s$ and $\pi_s(c) = \emptyset$ otherwise. We call a painting *connected* if each color forms a connected polygon: that is, $\bigcup_{s\in\Gamma} \pi_s(c)$ is a connected polygon for each color $c \in \{1, \dots, k\}$. For this definition two cells sharing only a corner are *not* considered connected. A (connected) painting is a disjoint-polygons representation of the hypergraphs captured by the colored grid. In the remainder we use painting to indicate a connected painting.

## 5.2 Characterizing colored grids with a painting

In this section we show how to test whether a 2-colored grid admits a painting and how to find a painting if one exists. As all red, blue, and white panels are fixed, finding a painting reduces to partitioning the purple cells such that resulting red and blue polygons are connected. We show that this connectivity is of key importance: if we can find suitable connections through the purple regions, then we can create a partition that results in a valid panel for each cell in the purple regions.

We capture the connectivity options for the red and blue polygon using two embedded graphs $G_r$ and $G_b$. We construct these graphs in three steps:

1. We identify and connect red (blue) regions that are adjacent along a purple region's boundary; we consider these as a single region in the remaining steps.
2. We remove any holes from the purple regions by inserting connections (Section 5.2.3).
3. Finally, we construct $G_r$ and $G_b$ using a gadget for purple regions (Sections 5.2.1 and 5.2.2).

For the first step, observe that consecutive (not necessarily distinct) adjacent regions of the same color can always be safely connected via the purple region's boundary without restricting the connectivity options for the other color (see Figure 5.2). After

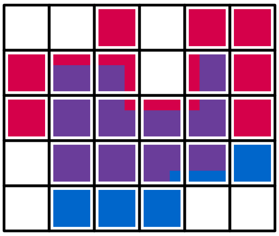


**Figure 5.2** Safe connections between adjacent same-color regions.

the first two steps, we represent the remaining red and blue regions as vertices in $G_r$ and $G_b$, respectively. Edges in $G_r$ and $G_b$ represent connection options through purple regions; intersections indicate a choice to connect either blue or red regions through (part of) a purple region. The gadget for purple regions with many adjacent red and blue regions also requires some additional vertices in these graphs. We prove that these graphs admit a simple characterization of 2-colored grids that admit a painting as captured in the theorem below.

5.2.1 **Theorem.** *A 2-colored grid* $\Gamma$ *admits a painting if and only if the corresponding graphs* $G_r$ *and* $G_b$ *are each other's exact duals: there is exactly one blue vertex in every red face and there is exactly one red vertex in every blue face.*

For explanatory reasons we start with the simplest case: purple regions with at most four neighbors and without holes (Section 5.2.1). Subsequently, we alleviate the assumption on the number of neighbors (Section 5.2.2) and permit holes in the purple regions by showing how to perform Step 2 (Section 5.2.3).

### 5.2.1 Simple purple regions

We assume that Step 1 has been performed and a purple region $P$ has no holes and at most four adjacent regions. The adjacent red and blue regions of $P$ form an ordered cyclic list as they appear along the boundary of $P$ and alternate in color (due to Step 1). Let $\kappa(P)$ denote the length of this list for $P$. $\kappa$ is even due to color alternation, and by assumption here $\kappa(P) \leq 4$. There can be duplicates in this list as the same red or blue region can touch $P$ multiple times.

Every purple region with $\kappa(P) = 2$ can be painted by creating a spanning tree on the centers of the panels of $P$ in one color and connecting it to the corresponding adjacent region of the same color. The rest of the panels is colored in the other color. We assume these are handled; what remains is to deal with the regions with $\kappa(P) = 4$.

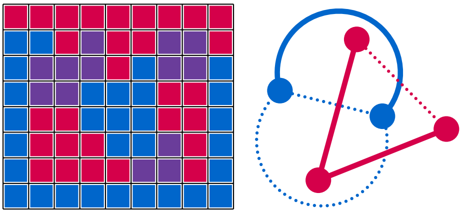


**Figure 5.3** 2-colored grid with 4 regions around each purple region and corresponding graphs $G_r$ and $G_b$.

For a purple region $P$ with $\kappa(P) = 4$ we create a red edge in $G_r$ and a blue edge in $G_b$ that intersect: the red edge connects the red vertices corresponding to the adjacent red regions; the blue edge connects the corresponding blue vertices. There may be multiple edges between two vertices (see Figure 5.3). If the same red or blue region touches the purple region twice, the edge is a self-loop. Every red or blue edge intersects exactly one blue or red edge respectively, and $G_r$ and $G_b$ are plane by construction. Using Lemma 5.2.2 we prove the exact characterization of graphs $G_r$ and $G_b$ of a 2-colored grid $\Gamma$ that admits a painting.

**5.2.2 Lemma ([38, 141]).** *Let $G$ be a plane graph, $G^*$ its dual and $T$ a spanning tree of $G$. Then $T^* = \{e^* \mid e \notin T\}$ is a spanning tree of $G^*$.*

**5.2.3 Lemma.** *A 2-colored grid $\Gamma$ in which each purple region $P$ has no holes and $\kappa(P) \leq 4$, admits a painting if and only if the corresponding graphs $G_r$ and $G_b$ are each other's exact duals.*

*Proof.* Suppose $\Gamma$ admits a painting $\Pi$. Then there exists a painting $\Pi'$ where for any purple region $P$, all consecutive neighboring regions of the same color are connected through the boundary of $P$. Let $n_r$ and $n_b$ denote the number of vertices in $G_r$ (red regions) and in $G_b$ (blue regions) respectively. We observe that the number of purple regions with $\kappa = 4$, the number of edges in $G_r$, the number of edges in $G_b$, and the number of intersections is the same, say, $e$. By construction, $n_b \geq f_r$ and $n_r \geq f_b$. The purple regions with $\kappa = 4$ can be painted to connect two adjacent red regions or two adjacent blue regions but never both. As there are $n_r$ red regions, at least $n_r - 1$ red edges are needed to connect all red vertices (regions). And similarly, at least $n_b - 1$ edges are needed to connect all blue vertices (regions). Thus, if $\Gamma$ admits painting $\Pi'$ then $e \geq n_r + n_b - 2$.

On the other hand, by Euler's formula $n_r - e + f_r = 2$ and $n_b - e + f_b = 2$. Combining these equations and $n_b \geq f_r$, $n_r \geq f_b$, we derive that $e \leq n_r + n_b - 2$. Thus, $e = n_r + n_b - 2$, and $n_b = f_r$ and $n_r = f_b$. As each red edge intersects exactly one blue edge and vice versa, graphs $G_r$ and $G_b$ are each other's duals.

For the other direction, assume that $G_r$ and $G_b$ are dual graphs. By Lemma 5.2.2 there exist two non-intersecting spanning trees of $G_r$ and $G_b$ that specify which adjacent regions of every purple region are to be connected to form connected red and blue polygons (see Figure 5.3). Given the connectivity information for every purple region it is straightforward to find the panels for all cells to obtain a connected painting: for every purple region draw the two spanning trees, and connect any cell that does not have yet a blue or a red piece to the blue or red polygon respectively with a connection without introducing any crossings. Thus, a 2-colored grid admits a painting if and only if $G_r$ and $G_b$ are dual graphs. □

## 5.2.2 Spiderweb gadgets

Let us now extend the result in the previous section by showing how to include purple regions with more than four adjacent regions. For every purple region $P$ with $\kappa(P) > 4$ we construct a *spiderweb* gadget and insert it into the graphs $G_r$ and $G_b$, such that an argument similar to Lemma 5.2.3 can be applied.

A spiderweb gadget $W$ of $P$ with $\kappa(P)/2 = k$ red and $k$ blue alternating adjacent regions consists of $\lfloor k/2 \rfloor + 1$ levels, labeled 0 (outermost) to $\lfloor k/2 \rfloor$ (innermost), see Figure 5.4. Each level, except 0 and $\lfloor k/2 \rfloor$, is a cycle of $k$ vertices. Level 0 has $k$ (blue) vertices without any edges between them, and the innermost level $\lfloor k/2 \rfloor$ consists of only a single vertex. The vertices of even levels are blue and labeled with even numbers from 0 to $2k-2$ clockwise. The vertices of odd levels are red and labeled with odd numbers 1 to $2k-1$ clockwise.

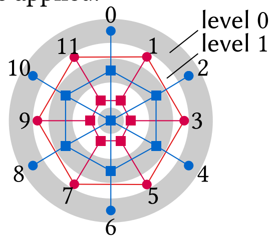


**Figure 5.4** Spiderweb gadget for $k$=6: three blue (0,2,4) and two red levels (1,3)

Each vertex of level $\ell$ with $2 \leq \ell < \lfloor k/2 \rfloor$ is connected to the vertex with the same label on level $\ell - 2$. The single vertex of level $\lfloor k/2 \rfloor$ is connected to all the vertices of level $\lfloor k/2 \rfloor - 2$. This gives us $2k$ paths starting from levels 0 and 1 to the two innermost levels. We call these paths *spokes*, and refer to them by the label of the corresponding vertices. We embed the two resulting connected components in such a way that they are each other's dual by making sure that we get a proper clockwise numbering on the vertices of the two outermost levels (see Figure 5.4). The vertices on levels 0 and 1 represent respectively the blue and red regions around the purple region $P$ and respect the adjacency order around $P$.

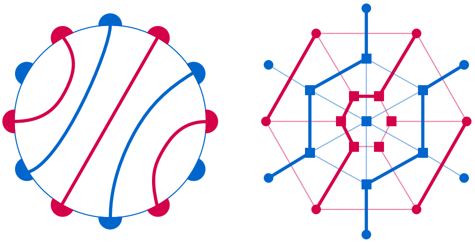


Figure 5.5 Topology of connections in a purple region and the corresponding bridging paths in a spiderweb gadget.

If a blue (or red) region touches $P$ multiple times, then the corresponding vertices on level 0 (or 1) map to the same region and are in fact one and the same vertex in $G_b$ (or $G_r$). All edges connected to this vertex are consistent with the topology of the nested neighboring regions of $P$; they intersect the same edges as they would when they were represented by multiple vertices.

To prove that all possible connections in $P$ which can occur in a painting $\Pi$ can be replicated in a spiderweb gadget $W$, we define *bridging paths*: let $u$ and $v$ be two vertices on level 0 in $W$ that represent two blue regions that are connected by a painting $\Pi$ through $P$. Assume that the clockwise distance from $u$ to $v$ is not greater than $k$, that is, if $u$ has label $x$ then $v$ has label $(x+2i) \mod 2k$ for some $1 \leq i \leq \lfloor k/2 \rfloor$. To connect $u$ and $v$ with a bridging path, we start from $u$, go to level $2\lfloor (i+1)/2 \rfloor$ along the spoke $x$, take a shortest path within the level $2\lfloor (i+1)/2 \rfloor$ from the vertex with label $x$ to the vertex with label $(x+2i) \mod 2k$, and move along the spoke $(x+2i) \mod 2k$ to vertex $v$. If there are two possible shortest paths, we take the clockwise path.

The same kind of path can be constructed for a pair of red vertices, but starting from level 1, going to level $2\lfloor i/2 \rfloor + 1$, and moving back to level 1. We now show that connecting different blue and red regions using bridging paths within the spiderweb gadgets results in blue trees and red trees, such that no pair of a blue and a red edge intersect (see Figure 5.5 for an example).

**5.2.4 Lemma.** *Consider a painting* $\Pi$ *in which two blue and two red regions, adjacent to a purple region* $P$*, are connected through* $P$*. The corresponding vertices in the spiderweb gadget* $W$ *of* $P$ *can be connected by non-intersecting bridging paths.*

*Proof.* Let $u$ and $v$ be two vertices that represent two blue regions adjacent to $P$ and connected through $P$ in painting $\Pi$. Let $u$ be on level 0 of spoke $x$ and $v$ be on level 0 of spoke $(x + 2i) \mod 2k$ for some $1 \leq i \leq \lfloor k/2 \rfloor$. We connect $u$ and $v$ by a bridging path as described above which goes to level $2\lfloor (i+1)/2 \rfloor$. Any connection of two red regions in $\Pi$ through the purple region $P$ cannot cross the connection between the regions that $u$ and $v$ represent in $\Pi$. This means that the corresponding vertices must both lie on the same side of the bridging path between $u$ and $v$. Denote these vertices by $u'$ and $v'$, and let $u'$ have label $y$ and $v'$ have label $(y + 2j) \mod 2k$ for by $1 \leq j \leq \lfloor k/2 \rfloor$. The bridging path goes to level $2\lfloor j/2 \rfloor + 1$. There can be several cases:

- When $u'$ and $v'$ lie in between $u$ and $v$ moving in the clockwise order, we have that $j < i$. Thus, the two bridging paths cannot intersect, as the path from $u'$ to $v'$ goes to level $2\lfloor j/2 \rfloor + 1 < 2\lfloor (i+1)/2 \rfloor$.
- When $u'$ and $v'$ lie in between $v$ and $u$ moving in the clockwise order, we have that either $j > i$ or the two bridging paths lie on the opposite sides of the vertex on the innermost level. In the later case the two bridging paths cannot intersect as they are separated by the innermost level, and in the former case the two bridging paths cannot intersect as the path from $u'$ to $v'$ goes to level $2\lfloor j/2 \rfloor + 1 > 2\lfloor (i+1)/2 \rfloor$.

□

With spiderweb gadgets and Lemma 5.2.4 we now strengthen Lemma 5.2.3 to the following lemma without a condition on $\kappa$.

5.2.5 **Lemma.** *A 2-colored grid $\Gamma$ in which each purple region has no holes admits a painting if and only if the corresponding $G_r$ and $G_b$ are each other's exact duals.*

*Proof.* The proof is similar to the proof of Lemma 5.2.3. Suppose $\Gamma$ admits a painting $\Pi$. Then there exists a painting $\Pi'$ where for any purple region $P$, all consecutive neighboring regions of the same color are connected through the boundary of $P$. We can find non-intersecting trees in every spiderweb gadget $W$ corresponding to a purple region $P$ that connect all pairs of vertices of levels 0 and 1 in the same way as $\Pi$. First, we create the bridging paths from Lemma 5.2.4 for every pair of regions of the same color connected in $\Pi$ through $P$. The bridging paths do not create cycles in $S$, thus, every vertex in $W$ that is still not connected to the levels 0 or 1 can be connected to them by growing non-intersecting spanning forests from the vertices on the levels 0 and 1.

For the other direction, assume that $G_r$ and $G_b$ are dual graphs. By Lemma 5.2.2 there exist two non-intersecting spanning trees of $G_r$ and $G_b$. These spanning trees provide the decisions of which adjacent regions of every purple region to connect,

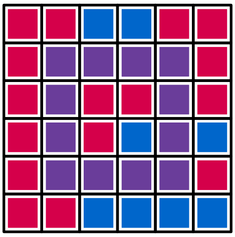


**Figure 5.6** An purple annulus with red and blue regions on both sides.

and the topology of the connections. Given the connectivity information, a painting can be constructed in a similar way to Lemma 5.2.3. Thus, a 2-colored grid admits a painting if and only if $G_r$ and $G_b$ are dual graphs. □

### 5.2.3 Purple regions with holes

We may also have purple regions with holes (see Figure 5.6). We show that the number of holes can be reduced without affecting the solvability. For simplicity of explanation we assume a region with a single hole (an annulus); regions with more holes can be reduced by considering only connections to the outer boundary.

Let $P$ be a purple annulus. Any painting subdivides $P$ into a number of colored simple components. Each component of color $c$ connects one or more regions of color $c$ on the boundary of $P$. The existence of a painting is defined only by the connectivity structure of these components. The connectivity of a component can be represented (transitively) using a set of non-intersecting simple paths (*connections*) each connecting two regions on the boundary. Let a *cross-annulus connection* $\gamma_x$ be a connection between a region $x_{in}$ on the inside of the annulus and a region $x_{out}$ on the outside of the annulus. A (connectivity) *structure* is a maximal set of (pairwise non-intersecting) connections in $P$ that can be extended to a valid painting. Let $C_S$ be the set of cross-annulus connections in a given structure $S$. We first assume the annulus is not degenerate, and thus red and blue regions exist both inside and outside the annulus. A degenerate annulus is a strictly simpler case having only cross-annulus connections of one color. The lemmas and proofs given below can be easily adapted for the degenerate case.

5.2.6 **Lemma.** *If a structure $S$ exists with two adjacent cross-annulus connections $\gamma_x$ and $\gamma_y$ of the same color, possibly separated by non-crossing connections, then there also exists a structure $S'$ where $C_{S'} = C_S \setminus \{\gamma_y\}$.*

*Proof.* As there are no cross-annulus connections between $\gamma_x$ and $\gamma_y$, these cross-annulus connections must be connected inside the annulus. As a result, after removing $\gamma_y$ we can reconnect the structure by introducing two non-crossing connections between $\gamma_x$ and the two regions that $\gamma_y$ connects. This does not change the connectivity of the structure, but removes the explicit cross-annulus connection $\gamma_y$ (see Figure 5.7). Note that there cannot be a connection between $\gamma_x$ and $\gamma_y$ outside the annulus as this would disconnect any blue regions in between. □

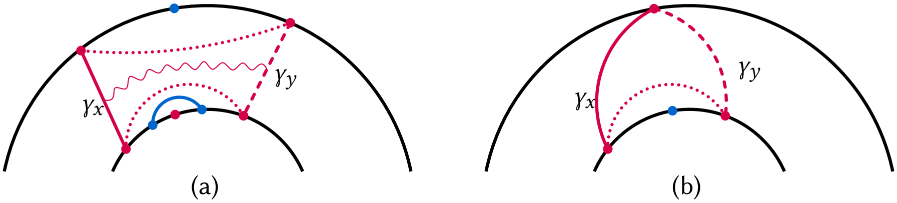


**Figure 5.7** (a) When there are two adjacent cross-annulus connections of the same color, $\gamma_x$ and $\gamma_y$, we can remove $\gamma_y$ (dashed) and keep the same connectivity using two non-crossing connections (dotted). Note that, due to having no cross-annulus connections between $\gamma_x$ and $\gamma_y$ they are implicitly connected (zigzagged line). (b) The analogous case, if there are no blue regions between $\gamma_x$ and $\gamma_y$ on one of the sides.

**5.2.7 Lemma.** *If there exists a structure $S$ with $|C_S| > 3$ and all cross-annulus connections are alternating in color, then there also exists a structure $S'$ with $|C_S| - 2$ cross-annulus connections.*

*Proof.* Let $\gamma_u$, $\gamma_v$, $\gamma_x$, and $\gamma_y$ be four consecutive cross-annulus connections. W.l.o.g., assume $\gamma_u$ and $\gamma_x$ are red and $\gamma_v$ and $\gamma_y$ are blue. We remove $\gamma_v$ and $\gamma_x$ from the structure separating both the red and blue into two components. For both colors, one component remains connected to the other cross-annulus connection $\gamma_u$, respectively $\gamma_y$. The disconnected components cannot both be on the outside (inside) of the annulus. If this were the case, the connection $\gamma_u$ to $\gamma_x$ must be connected through $x_{in}$, and $\gamma_v$ to $\gamma_y$ through $v_{in}$. However, as there is no cross-annulus connection between $\gamma_u$ and $\gamma_y$, any connection from $\gamma_u$ to $x_{in}$ separates $\gamma_y$ and $v_{in}$. Hence, both connections cannot exist at the same time. The red and blue disconnected components are thus on different sides of the annulus and we connect them to $\gamma_u$, respectively $\gamma_y$, without mutually interfering (see Figure 5.8). □

**5.2.8 Corollary.** *If a structure exists, then a structure also exists that has exactly one red and one blue connection across each annulus.*

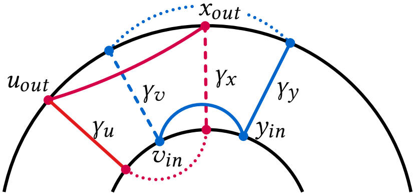


**Figure 5.8** By adding edges $(v_{in}, y_{in})$ and $(u_{out}, x_{out})$ we reconnect the disconnected subpolygons formed by removing cross-annulus connections $\gamma_v$ and $\gamma_x$.

*Proof.* By repeated application of Lemma 5.2.7, we can find a structure with at most three cross-annulus connections. By Lemma 5.2.6, we can reduce it further to a structure with two cross-annulus connections. □

**5.2.9 Lemma.** *If a structure exists, then there also exists a structure with exactly one red and one blue cross-annulus connection starting from any two regions on the inner annulus and connecting to any two regions on the outer annulus.*

*Proof.* Let an *interval* be a maximal arc of the same color on the boundary. By Corollary 5.2.8 we know there exists a structure with exactly one red and one blue connection across the annulus. Let $\gamma_x$ be the blue cross-annulus connection and $\gamma_y$ the red cross-annulus connection. We show that each of the endpoints of the cross-annulus connection can freely be moved. W.l.o.g., assume that $\gamma_x$ is not counterclockwise adjacent to $\gamma_y$ on the outside of the annulus. Let $k_{out}$, $l_{out}$, and $m_{out}$ be three intervals in clockwise order on the outer boundary of the annulus. We say a clockwise connection through the annulus from $k_{out}$ to $m_{out}$ *covers* $l_{out}$.

Let $b_{out}$ be the blue interval that is counterclockwise adjacent to $y_{out}$ and $r_{out}$ the red interval that is counterclockwise adjacent to $b_{out}$. Interval $b_{out}$ may have several incoming blue connections that cover $r_{out}$ (see Figure 5.9(a)). We can rewire the blue connections inside the annulus to connect the blue intervals in sorted order around the annulus resulting in only one blue connection $\gamma_b$ that covers $r_{out}$. Similarly, we can also rewire the red connections covering $r_{out}$ and ending at $y_{out}$ to ensure that only one red connection $\gamma_r$ covers $r_{out}$.

Remove $\gamma_y$ and insert a new red cross-annulus connection $\gamma_z = (y_{in}, r_{out})$. The connection $\gamma_z$ can intersect only $\gamma_r$ and $\gamma_b$. Removing $\gamma_r$ results in two red components, one of which contains $\gamma_z$. Assume w.l.o.g. that $y_{out}$ is in the same connected component as $\gamma_z$. As $\gamma_r$ intersected $\gamma_z$, the disconnected component can be connected to $\gamma_z$ while intersecting only $\gamma_b$ (see Figure 5.9(b)).

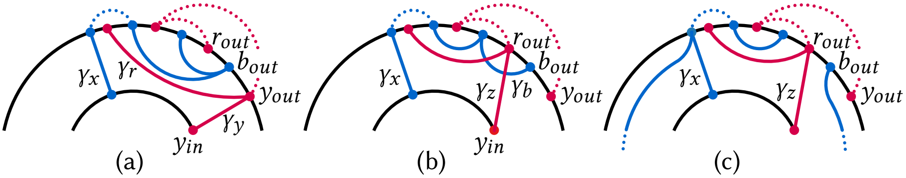


**Figure 5.9** (a) Initial configuration with several connections covering $r_{out}$. (b) Rerouting the blue connections, introducing $\gamma_z$, and rerouting the intersecting red connection leaves only one intersecting (blue) connection. (c) The blue disconnected component cannot be covered by the new red connection, we reconnect it to $\gamma_x$.

Removing $\gamma_b$ results in two blue components, one of which contains $\gamma_x$. We prove that $b_{out}$ must be part of the blue component not containing $\gamma_x$. Assume to the contrary that $b_{out}$ is still connected to $\gamma_x$. Interval $r_{out}$ must be connected to $y_{out}$ outside of the annulus as there was only one red cross-annulus connection and $\gamma_b$ blocked any connection through the inside of the annulus. Similarly, interval $b_{out}$ must have been connected through the outside of the annulus, as it is separated from any other region inside the annulus by $\gamma_y$ and $\gamma_z$. However, they cannot both be connected through the outside of the annulus, as the connection $r_{out}$ to $y_{out}$ separates $b_{out}$ and $x_{out}$ on the outside of the annulus. Contradiction.

Therefore, we can safely reconnect the disconnected blue component through the annulus to $\gamma_x$ (see Figure 5.9(c)). Repeatedly moving the end-point of one of the cross-annulus connections allows the creation of any configuration of the two red and blue cross-annulus connections without invalidating the structure. □

Lemma 5.2.9 implies that we can cut the annulus open to reduce the number of holes of a purple region by one without changing the solvability of the problem. Together with Lemma 5.2.5 this then implies Theorem 5.2.1.

## 5.3 Optimizing panels

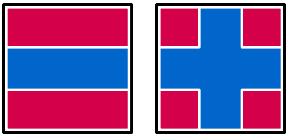


**Figure 5.10** Panels with complexity 3 and 5.

As shown, not all colored grids admit a painting. Here we investigate the design of the panels themselves assuming that some painting exists. To this end, we define the complexity of a panel as the number of *pieces* of maximal red and blue areas in the panel, see Figure 5.10. The complexity of a painting is the maximal complexity of any of its panels. A $t$-panel ($t$-painting) has complexity $t$. Assuming some painting exists, we prove in this section that a 5-painting exists in general and that even a 2-painting exists if there are no white cells.

### 5.3.1 Ensuring a 5-painting

We prove here that a 5-painting can always be realized. To this end, we show that a valid painting for a colored grid can be redrawn to include no more than three colored intervals along each side of all panels.

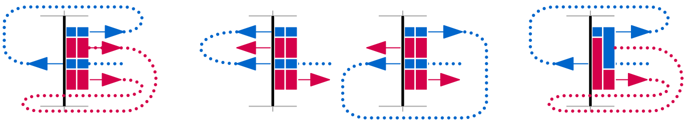
**5.3.1 Lemma.** *If a 2-colored grid admits a painting, then it admits a painting where each panel $\pi$ has at most 3 intervals of alternating red and blue along each side.*

*Proof.* Without loss of generality, assume that a panel $\pi$ has at least 4 intervals of alternating red and blue on the left-side of $\pi$. We consider the top-most four intervals that are inside $\pi$ and adjacent to the left-side of $\pi$ and w.l.o.g. we assume these intervals are ordered blue, red, blue, red. As the painting is valid, both blue as well as the red intervals are connected in the painting by exactly one path, since there can be no cycles. For each interval we identify whether the path exiting or entering $\pi$ connects to the other interval of the same color (see Figure 5.11(a)). It cannot be the case that the red and blue path both leave or exit $\pi$ in the same direction for the middle two intervals (see Figure 5.11(b)). Assume w.l.o.g. that the middle blue connecting path exits $\pi$. Any path connecting the blue intervals must separate the left side of the top red interval from the bottom red interval. Hence, the connecting path from the middle red interval cannot also exit $\pi$.

To reduce the number of intervals, we recolor the interval of the color whose connecting path exits $\pi$ (blue in Figure 5.11(a)) to the other color. To keep the blue polygon connected and remove the newly created red cycle, we move the other blue interval a small distance inside $\pi$ and stretch it over the middle red interval. (see Figure 5.11(c)). This reduces the number of intervals on the boundary of $\pi$ by two and can be repeated as necessary without affecting the validity of the solution. □

**5.3.2 Theorem.** *If a partially 2-colored grid admits a painting, then it admits a 5-painting.*

*Proof.* By Lemma 5.3.1 there are at most three alternating colored intervals along each side of $\pi$. If a red and blue interval meet in a corner, we extend one in $\pi$ around the corner to get four intervals and use Lemma 5.3.1 to reduce it back to at most three. If we have more than five pieces, a piece that has only one interval in $\pi$ can be removed while maintaining a painting. Each remaining piece connects at least



(a) (b) (c)

**Figure 5.11** (a) The side of a panel $\pi$ with 4 intervals. (b) The two middle directions cannot be the same, as we cannot connect them with non-intersecting paths. (c) Shortcutting inside $\pi$ reduces the number of intervals while maintaining a painting.

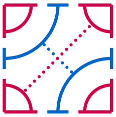
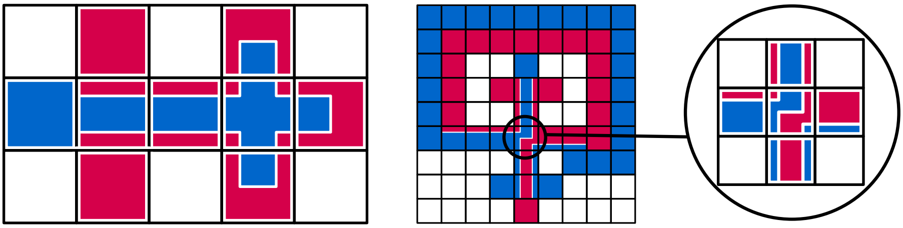


**Figure 5.12** A panel with six pieces can always be reduced to have five pieces, using either of the dotted lines.

two intervals: with $k$ intervals, the number of pieces is at most $\lfloor k/2 \rfloor$. A 6-panel thus requires 12 intervals: four equal-color (red) corners and a middle interval (blue) along each side. This enforces two pieces between the blue intervals, and one in each corner. However, we can now reduce the number of pieces to five, connecting either two blue pieces or two red corners (Figure 5.12). □

This bound is tight as a 5-panel may be required when the grid includes white cells (Figure 5.13). A 5-panel with at least two pieces of each color is never required—though such a 4-panel may be necessary (Figure 5.13(b)). The above proof implies that there is only one option to create such a 5-panel: it has only two ways to connect the two blue pieces; both can be simplified to a 4-panel (Figure 5.14).



(a) (b)

**Figure 5.13** Examples requiring complex panels. (a) A colored grid requiring a 5-panel. (b) A colored grid requiring a 4-panel with two pieces of both colors in the same cell (right) .

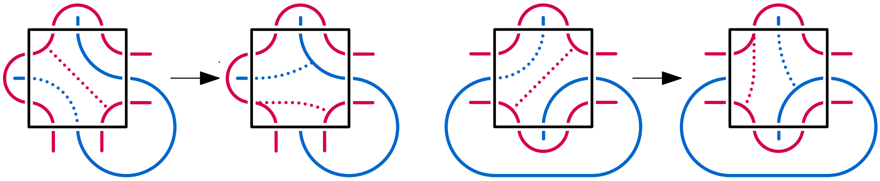


**Figure 5.14** There are two configurations for a 5-panel where both colors have at least two pieces. Both possible configuration can be simplified to a 4-panel.

## 5.3.2 Ensuring a 2-painting

We show that a fully 2-colored grid (rectangular and without white cells) even admits a 2-painting provided it admits any painting. As an intermediate step, we first prove that a painting exists that uses only one blue piece in any panel.

**5.3.3 Lemma.** *If a fully 2-colored grid admits a painting, then it admits a painting in which each panel has at most one blue piece.*

*Proof.* Let $\Pi$ be a painting admitted by a fully 2-colored grid $\Gamma$. Any fully red or blue panel in $\Pi$ trivially satisfies our lemma. Hence, we consider a purple region $P$ and show how to repanel it to ensure each panel has exactly one blue piece. Let $\Pi'$ be the repainted solution which is identical to $\Pi$ except for the repainted cells in $P$.

As there are no white cells, the blue and red intervals along the boundary of $P$ must alternate. Exceptions are the convex corners of $P$ where there may be two blue cells adjacent to the same purple cell separated by a red or purple cell (see Figure 5.15(a) top-left). As $\Gamma$ admits a painting, these blue cells must be connected through $P$ as a connection through the outside would isolate the red/purple corner cell from $P$. We refer to these connections through $P$ as the blue corners of $P$. At the outer boundary of the grid the same can happen and we can treat them similarly. Note that, although the connection may now span multiple cells it passes through each at most once.

We construct a new painting for $P$ as follows. First we place a small blue rectangle at the center of every panel in $P$. We then reconnect the blue corners of $P$ by filling the appropriate corners from this center rectangle. On these centers we build a spanning forest by connecting to the centers of adjacent cells. This spanning forest is built such that each tree in the forest is rooted in a distinct blue interval (see Figure 5.15(b)). By construction, all red pieces in $P$ forms a single polygon and thus red is connected. If there was only one blue polygon in $P$, then we are done as blue is connected and forms a single piece inside every cell in $P$.

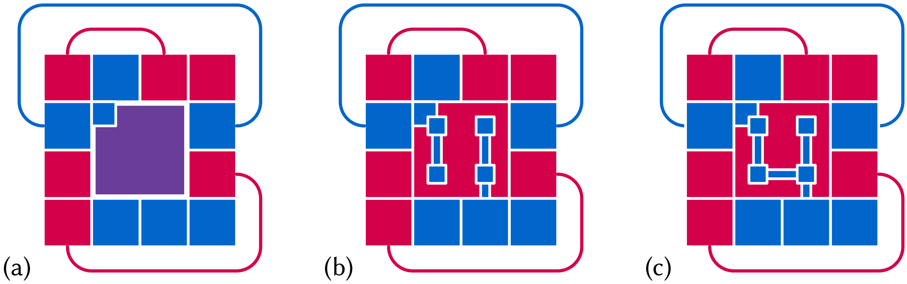


**Figure 5.15** A purple region of four cells that has to be modified, including the direst neighborhood and topological connections in the original painting Π outside $P$. (a) The top-left blue corner must be present. (b) Construction of the spanning forest. (c) Connecting two blue pieces breaks a cycle in the red polygon.

Assume there at least two blue polygons in $P$. Since we know that a painting exists, only a red cycle can be broken when we connect two blue polygons. This is most easily seen in the constructed graphs $G_b$ and $G_r$ (see Theorem 5.2.1): our constructed forest maps to a forest in $G_b$ and by Lemma 5.2.2 we can arbitrarily complete this into a tree while ensuring that $G_r$ has a tree as well. Hence, we can pick any two polygons that have adjacent cells in $P$ and connect them (see Figure 5.15(c)). This reduces the number of blue polygons by one while keeping the red polygon connected, as the connection only ever cuts a red cycle. We repeat the above for every purple region to ensure the eventual painting has a single blue piece in every panel. □

The above construction relies on the alternation of the blue and red intervals along the boundary of $P$. As there are no white cells we can guarantee this alternating pattern. Indeed, the higher complexity with white cells is caused by long connections along a purple region's boundary that are needed to achieve this alternating pattern for a partially colored grid (e.g., Figure 5.13).

**5.3.4 Theorem.** *If a fully* 2*-colored grid admits a painting, then it admits a* 2*-painting.*

*Proof.* If there is a solution for a fully 2-colored grid Γ, then by Lemma 5.3.3 there is a solution Π where every panel in a purple region only has a single blue piece. That is, every panel only has red along the sides or corners. All red pieces must include at least one corner by construction of Lemma 5.3.3. If a panel has a red piece that only connects to one neighboring panel then we retract it such that it is only adjacent to a single corner. While a panel has more than one red piece we can fully remove any such red piece from the panel.

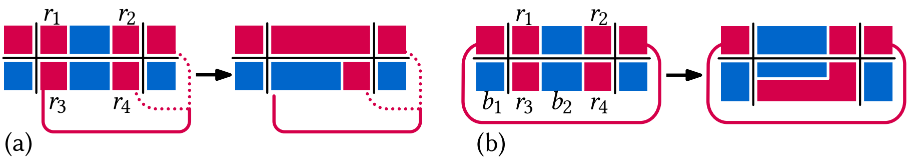


**Figure 5.16** Reducing panel complexity when there are two red corners along the same panel side. (a) The corners are connected via adjacent (or the same) sides of the panel: connect $r_1$ and $r_2$, and recolor $r_3$ to blue. (b) The corners are connected via opposite sides: recolor $r_1$ to blue and connect $r_3$ and $r_4$ as well as $b_1$ and $b_2$.

Next we remove unnecessary red pieces around corner points from purple panels that have at least two distinct red pieces. If all four cells around a corner point of a purple panel have a red corner adjacent to the corner point, then we can color the red corner of the purple panel blue without influencing the connectivity. After repeated application of the above, any panel with multiple red components is in one of four cases:

1. There are two red corners $r_1$ and $r_2$ on the same side of the panel. The connecting path exists the current panel via the same side and enters either on the same or adjacent side. (see Figure 5.16(a)).
2. There are two red corners $r_1$ and $r_2$ on the same side of the panel. The connecting path exits the panel via opposite sides of the panel (see Figure 5.16(b)). The blue piece connects only downwards in the panel below.
3. There are two red corners $r_1$ and $r_2$ that do not share a common side of the panel. In this case the other corners are blue, otherwise one of the two previous cases applies (see Figure 5.17). Furthermore, either $p_1$ or $p_2$ is blue.
4. There are two red corners $r_1$ and $r_2$ that do not share a common side of the panel. Furthermore, both $p_1$ and $p_2$ are red.

We can reduce the complexity of each panel the following reduction rules:

1. Connect $r_1$ and $r_2$, and remove $r_3$ to break the red cycle (see Figure 5.16(a)).
2. Connect $r_3$ and $r_4$, and connect $b_1$ and $b_2$ between $r_1$ and $r_3$. We remove $r_1$ as it has become useless by connecting $b_1$ and $b_2$. Similarly, we fully color the rest of the bottom panel red (see Figure 5.16(b)). Note that the only option is for the blue piece in the bottom panel to connect solely to the panel below it. Assume that it also connected to the left panel. Then the left red piece must

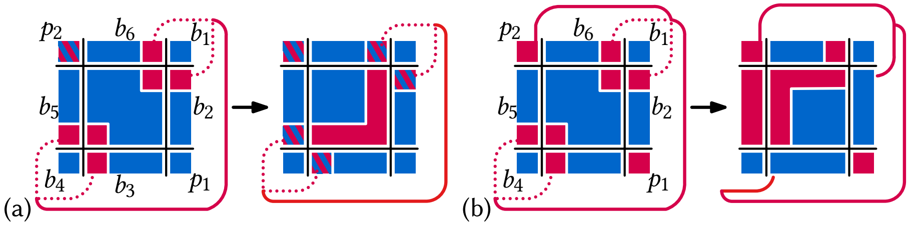


**Figure 5.17** Two diagonally positioned red corners. The complexity of the panel can be reduced by introducing a red $L$-shape that connects all the red. (a) Reducing complexity if either $p_1$ or $p_2$ was blue. (b) Reducing complexity if both $p_1$ and $p_2$ were red.

also connect to the left panel. When this red piece connects two different sides of this panel then it envelopes the blue corner and this panel cannot have anymore blue below. This contradicts that there was a solution. Thus, if the red piece ended in this panel it should include at least one corner, which it does not. Hence, this situation can also not occur.

3. If either $p_1$ or $p_2$ is blue, we connect $r_1$ and $r_2$ along this respective side of the panel (see Figure 5.17(a)). We connect either $b_1$ or $b_4$ to an adjacent blue polygon to break the red cycle and create a single blue polygon again.

4. When both $p_1$ and $p_2$ are red, we connect $r_1$ and $r_2$ along the side of the panel that is on the opposite side of the corner that the connection between both red pieces encloses (we pass $p_2$ in Figure 5.17(b)). This results in three blue polygons. Assume w.l.o.g. that the red corners are connected via a path leaving through the bottom and entering from the right. We join the blue polygons together by connecting $b_1$ and $b_2$, and $b_3$ and $b_4$, separating both sides of the connecting path. We also recolor the border along the side of the adjacent panel that is not enclosed by the red connection ($b_5$ in this case) to ensure connectivity of the red.

Repeated application of the above reduction rules interlaced with the reduction of the number of red pieces in a panel must result in a 2-painting. Every panel still has a single blue piece as no reduction rule increases the number of blue pieces in a panel. Furthermore, as none of the above rules applies anymore every panel has a single red polygon as well. Thus, the painting is a 2-painting. □

## 5.4 Conclusion

We took the first steps towards investigating a disjoint-polygons representation for visualizing sets (hypergraphs). The disjoint-polygon representation allows for sets to be displayed without any overlap on the nodes, possibly resulting in a more coherent set visualization. We investigated the variant where we have 2 hyperedges, and thus also 2 colors. Each element of the hyperedge is positioned as a cell in a (unit) grid, for example coming from a grid map. We showed how to test whether a disjoint-polygons representation exists for a given 2-colored grid. Moreover, we proved that if such a representation is possible, then we can also bound the complexity of the corresponding "panels" (the coloring of a single cell). Each panel requires at most five colored pieces, and even only two pieces are sufficient when no white cells are present in the grid.

There are myriad options for further exploration. We have not touched upon variants with more colors: does our approach readily generalize? Considering the restrictions already present in the studied 2-color variant it seems likely that many practical instances do not admit a painting. It would therefore be interesting to also study the problem of minimizing the number of polygons of the same color. Additionally, we could allow for some rearrangement of the elements which would make the method more practically applicable at the cost of weakening the spatial coherence. Finally, we may consider the situation where some cells have no assigned set of colors but may be painted using any subset of the colors. Given enough such cells we could encode more classes of graphs.

# Chapter 6

# Conclusion

In this thesis we studied algorithms for coherent rectangular visualizations. Coherence requires that relations between data items need to be visually represented, and that visible relations are present in the data. A coherent visualization thus prevents false patterns from emerging in the visualization, and preserves patterns that exists in the data. We investigated coherence for four different data types: *(1)* time-varying hierarchical data, *(2)* uncertain hierarchical data, *(3)* geospatial data, and *(4)* spatial set data. Below, we summarize our results and provide suggestions for future work.

## 6.1 Main results

For time-varying hierarchical data we introduced the Local Moves algorithm to maintain temporal coherence in treemaps. The algorithm uses local moves, small local modifications to the treemap. By limiting the number of local moves, the trade-off between temporal coherence and visual quality can be controlled. In contrast to existing treemapping algorithms which can reach only sliceable layouts, the full range of layouts can be explored using local moves. To verify the efficacy of the Local Moves algorithm as well as existing treemapping algorithms, we performed an extensive quantitative evaluation of rectangular treemapping algorithms for time-varying hierarchical data. To better measure the temporal coherence for this evaluation, we introduced a new methodology based on baseline treemaps to take the amount of change in the input data into account. Furthermore, we proposed a novel classification scheme for time-varying datasets using four representative features. We experimentally validated this classification, and used it to analyze the relative performance of treemapping algorithms across the features on both temporal coherence and visual quality.

For uncertain hierarchical data we introduced hierarchical uncertainty masks for treemaps to obtain coherence with the relation between data and uncertainty. The resulting Uncertainty Treemaps visualize both numerical values and their associated uncertainty simultaneously using area. Furthermore, we showed how to adapt existing treemapping algorithms to support uncertainty masks. To this end, we defined a quality measure for uncertainty masks to steer and evaluate these algorithms. We demonstrated the quality of the adapted treemapping algorithms through a computational experiment on real-world datasets.

For complex geospatial data we introduced a simple fully-automated 3-step pipeline to compute grid maps that are coherent with the spatial dimension. We observed that any input can be decomposed into simple cases which can be solved well by existing grid-map algorithms. The coherent solutions for the resulting simple subproblems can then be combined into a coherent overall solution. Each step of our pipeline is a well-studied problem: shape decomposition based on salient features, tile-based Mosaic Cartograms, and point-set matching. Our pipeline is a seamless composition of existing techniques for these problems and results in high-quality and coherent grid maps. We demonstrated the efficacy of our approach on various complex datasets, and compared it to the state-of-the-art.

Finally, for spatial set data we introduced a new visual representation to coherently visualize the relations between sets. The sets are represented by disjoint colored polygons, and the data items are represented by simple geometric shapes. As a first exploration of the feasibility of this visual representation, we assumed that the shapes are rectangles positioned on a grid, and that there are only two sets. Under these assumptions, we derived a necessary and sufficient condition to efficiently recognize whether we can visualize each set with a single connected polygon that is disjoint from the other polygon. Additionally, we showed that, if both sets can be visualized with such disjoint polygons, then the required visual complexity of the polygon in each rectangle is bounded by a small constant.

## 6.2 Future work

An obvious direction for future work is to investigate coherence for more data types. One example of such a data type, is temporal spatial set data, where the spatial position of the elements changes over time. For this type of data the visualization needs to not only coherently visualize the relations between the sets, but additionally it needs to ensure coherence with the temporal dimension. The challenge then lies with maintaining a suitable, natural visual representation, that avoids unwanted overlap between hyperedges as the elements move, while keeping the drawing stable to obtain temporal coherence. Once we understand the desired representation, we may again be able to define a set of "local moves" to control the trade-off between stability and other quality measures for this type of data.

A second direction for future work is to investigate coherence for different visualization techniques. For example, geospatial data can be represented by a multitude of different visualization techniques; one such technique is a Demers cartogram. In a Demers cartogram, each spatial element is represented by a square, whose area encodes a numerical value. These squares need to be disjoint and thus the underlying spatial dimension needs to be distorted in a way that is coherent with the spatial relations between the elements. Current techniques to generate Demers Cartograms are not always able to preserve coherence with the global shape. It would be interesting to see whether Demers cartograms admit a solution similar to our pipeline for coherent grid maps, where we decompose the input shape into simpler regions and combine the solutions for these regions into a coherent overall solution.

There are still interesting open problems for the data types and visualizations presented in this thesis as well. We discussed specific directions for future work in each chapter; below we discuss two overarching challenges.

**Measuring coherence** To compare and evaluate coherent visualizations, we need to be able to accurately measure the coherence of a visualization. Typically, coherence does not depend only on a single visual variable, but on more general structures in the visualization. Hence, it is challenging to base coherence measures on perceptual research.

Taking geospatial data as an example, it is difficult to measure the spatial coherence with respect to the input shape. Common shape similarity measures are not suitable for handling distortion, which is a necessity in obtaining a high-quality grid map. Yet, not all distortion is equal; it must be applied effectively and in a structured way as to still have a sense of local scale for salient features, even though the larger structures are distorted. We thus need a coherence measure that takes both the local structures and the similarity to the input shape into account.

Complicating the issue of evaluating a coherent visualization further, is that their efficacy depends on how accurate individual items are perceived. In treemaps for example, the Slice-and-Dice algorithm is coherent with the temporal dimension, which intuitively should make it suitable for time-varying data. However, the area of the rectangles in a Slice-and-Dice treemap are hard to assess due to the high aspect ratios. Therefore, the Slice-And-Dice treemap is not likely to perform well on tasks for time-varying data. There is thus a need for studies that investigate the interplay between coherence and other quality measures. Such a study could deliver important insights as to where on the Pareto-front of the different quality measures a visualization should be.

**Controlling trade-offs** In general, we often encounter a trade-off between the coherence and other quality metrics of a visualization. Improving the quality on one metric, then reduces the quality of another metric. Depending on the exact task, different metrics can be deemed more important. Hence, the visualization may need to be in different spots on the Pareto-front of quality metrics for different tasks. We have shown that we can control the trade-off between the visual quality and the temporal coherence in treemaps. It would be interesting to see whether a similar result is possible for the other data types.

Often, there also exists a trade-off between the running time and the quality metrics of a visualization. For time-varying hierarchical data in treemaps, we can increase the visual quality by allowing the algorithm to explore more layouts, at the cost of increased running time. However, this will also decrease the coherence with the temporal dimension. It is an interesting open problem whether this trade-off between the running time, the coherence, and other quality measures can be fully controlled for both this data type in particular, as well as for other data types in general.

# Bibliography

**[1]** B. Alper, N. Riche, G. Ramos, and M. Czerwinski. “Design study of LineSets, a novel set visualization technique”. In: *IEEE transactions on visualization and computer graphics* 17.12 (2011), pp. 2259–2267.

**[2]** B. Alsallakh, W. Aigner, S. Miksch, and H. Hauser. “Radial Sets: Interactive visual analysis of large overlapping sets”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2496–2505.

**[3]** B. Alsallakh, L. Micallef, W. Aigner, H. Hauser, S. Miksch, and P. Rodgers. “The State of the Art of Set Visualization”. In: *Computer Graphics Forum* 35.1 (2016), pp. 234–260.

**[4]** D. Archambault and H. C. Purchase. “Mental map preservation helps user orientation in dynamic graphs”. In: *International Symposium on Graph Drawing*. Springer. 2012, pp. 475–486.

**[5]** *ATP Tennis Rankings*. `https://github.com/JeffSackmann/tennis_atp`. accessed 03-07-2018.

**[6]** A. S. Bair, D. H. House, and C. Ware. “Factors influencing the choice of projection textures for displaying layered surfaces”. In: *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization* (2009), p. 101.

**[7]** M. Balzer, O. Deussen, and C. Lewerentz. “Voronoi Treemaps for the Visualization of Software Metrics”. In: *Proceedings of the ACM Symposium on Software Visualization.* 2005, pp. 165–172.

**[8]** M. Baron. “A note on the historical development of logic diagrams: Leibniz, Euler and Venn”. In: *Mathematical Gazette* 53.384 (1969), pp. 113–125.

**[9]** B. B. Bederson, B. Shneiderman, and M. Wattenberg. “Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies”. In: *ACM Transactions on Graphics* 21.4 (2002), pp. 833–854.

**[10]** M. de Berg, B. Speckmann, and V. van der Weele. “Treemaps with bounded aspect ratio”. In: *Computational Geometry: Theory and Applications* 47.6 (2014), pp. 683–693.

**[11]** B. Berkowitz and L. Gamio. *What you need to know about the measles outbreak.* Accessed April 2020. Feb. 2015. URL: `https://www.washingtonpost.com/graphics/health/how-fast-does-measles-spread/`.

**[12]** J. Bertin. *Semiology of graphics: Diagrams, networks, maps*. University of Wisconsin Press, 1983.

**[13]** I. Biederman. "Recognition-by-Components: A Theory of Human Image Understanding". In: *Psychological Review* 94.2 (1987), pp. 115–147.

**[14]** C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

**[15]** J. L. Blaha. *Standard Errors in the Consumer Expenditure Survey*. `https://www.bls.gov/cex/anthology/csxanth5.pdf`. 2003.

**[16]** P. Blickle and S. Venohr. *Dürfen wir vorstellen: Deutschlands Muslime*. Accessed April 2020. Jan. 2015. URL: `http://www.zeit.de/gesellschaft/2015-01/islam-muslime-in-deutschland`.

**[17]** H. Blum. "A transformation for extracting new descriptors of shape". In: *Models for the perception of speech and visual form* 19.5 (1967), pp. 362–380.

**[18]** S. A. Boorman and D. C. Oliviera. "Metrics on spaces of finite trees". In: *Journal of Mathematical Psychology*. Vol. 10. 1973, pp. 26–59.

**[19]** N. Boukhelifa, A. Bezerianos, T. Isenberg, and J.-D. Fekete. "Evaluating Sketchiness as a Visual Variable for the Depiction of Qualitative Uncertainty". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2769–2778.

**[20]** Q. W. Bouts, I. Kostitsyna, M. van Kreveld, W. Meulemans, W. Sonke, and K. Verbeek. "Mapping Polygons to the Grid with Small Hausdorff and Fréchet Distance". In: *Proceedings of the 24th Annual European Symposium on Algorithms*. Vol. 57. 2016, 22:1–22:16.

**[21]** F.-J. Brandenburg. "On the complexity of optimal drawings of graphs". In: *Proceedings of the Graph-Theoretic Concepts in Computer Science*. Vol. 411. 1989, pp. 166–180.

**[22]** K. Brodlie, R. Allendes Osorio, and A. Lopes. "A Review of Uncertainty in Data Visualization". In: *Expanding the Frontiers of Visual Analytics and Visualization*. Springer, 2012, pp. 81–109.

**[23]** M. Bruls, K. Huizing, and J. J. van Wijk. "Squarified Treemaps". In: *Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*. 2000, pp. 33–42.

**[24]** K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. "On Planar Supports for Hypergraphs". In: *Journal of Graph Algorithms and Applications* 15.4 (2011), pp. 533–549.

**[25]** R. G. Cano, K. Buchin, T. Castermans, A. Pieterse, W. Sonke, and Bettina. "Mosaic Drawings and Cartograms". In: *Computer Graphics Forum* 34.3 (2015), pp. 361–370.

**[26]** B. Casselman and A. McCann. *Where Your State Gets Its Money.* Accessed April 2020. Apr. 2015. URL: `http://fivethirtyeight.com/features/where-your-state-gets-its-money/`.

**[27]** M. Chen, S. Walton, K. Berger, J. Thiyagalingam, B. Duffy, H. Fang, C. Holloway, and A. E. Trefethen. "Visual multiplexing". In: *Computer Graphics Forum* 33.3 (2014), pp. 241–250.

**[28]** Y. Chen, X. Du, and X. Yuan. "Ordered small multiple treemaps for visualizing time-varying hierarchical pesticide residue data". In: *Visual Computer* 33.6 (2017), pp. 1073–1084.

**[29]** F. Chin, J. Snoeyink, and C. A. Wang. "Finding the medial axis of a simple polygon in linear time". In: *Discrete & Computational Geometry* 21.3 (1999), pp. 405–420.

**[30]** W. S. Cleveland and R. McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods". In: *Journal of the American Statistical Association* 79.387 (1984), pp. 531–554.

**[31]** S. Cohen and L. Guibas. "The Earth Mover's Distance under transformation sets". In: *Proceedings of the IEEE International Conference on Computer Vision.* Vol. 2. 1999, pp. 1076–1083.

**[32]** C. Collins, G. Penn, and S. Carpendale. "Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1009–1016.

**[33]** M. Correll and M. Gleicher. "Error bars considered harmful: Exploring alternate encodings for mean and error". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2142–2151.

**[34]** D. DeBelius. *Let's Tesselate: Hexagons For Tile Grid Maps.* Accessed April 2020. May 2015. URL: `http://blog.apps.npr.org/2015/05/11/hex-tile-maps.html`.

**[35]** T. C. van Dijk and J.-H. Haunert. "Interactive focus maps using least-squares optimization". In: *International Journal of Geographical Information Science* 28.10 (2014), pp. 2052–2075.

**[36]** K. Dinkla, M. van Kreveld, B. Speckmann, and M. Westenberg. "Kelp Diagrams: Point set membership visualization". In: *Computer Graphics Forum* 31.3pt1 (2012), pp. 875–884.

**[37]** B. Engdahl. “Ordered and unordered treemap algorithms and their applications on handheld devices”. In: *Master’s degree project, Department of Numerical Analysis and Computer Science, Stockholm Royal Institute of Technology, SE-100* 44 (2005).

**[38]** D. Eppstein, G. Italiano, R. Tamassia, R. Tarjan, J. Westbrook, and M. Yung. “Maintenance of a minimum spanning forest in a dynamic plane graph”. In: *Journal of Algorithms* 13.1 (1992), pp. 33–54.

**[39]** D. Eppstein, M. van Kreveld, B. Speckmann, and F. Staals. “Improved Grid Map Layout by Point Set Matching”. In: *International Journal on Computational Geometry Applications* 25.2 (2015), pp. 101–122.

**[40]** D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek. “Area-universal and constrained rectangular layouts”. In: *SIAM Journal on Computing* 41.3 (2012), pp. 537–564.

**[41]** M. Espadoto, R. Martins, A. Kerren, N. Hirata, and A. Telea. “Towards a Quantitative Survey of Dimension Reduction Techniques”. In: *IEEE Transactions on Visualization and Computer Graphics* (2019).

**[42]** P. F. Fisher. “Models of uncertainty in spatial data”. In: *Geographical information systems* 1 (1999), pp. 191–205.

**[43]** B. Fong. *How to Make Tile Grid Maps in Tableau.* Accessed April 2020. Jan. 2016. URL: `https://public.tableau.com/s/blog/2016/01/how-make-tile-grid-maps-tableau`.

**[44]** M. T. Gastner and M. E. J. Newman. “Diffusion-based method for producing density-equalizing maps”. In: *Proceedings of the National Academy of Sciences* 101.20 (2004), pp. 7499–7504.

**[45]** M. T. Gastner, V. Seguy, and P. More. “Fast flow-based algorithm for creating density-equalizing map projections”. In: *Proceedings of the National Academy of Sciences* 115.10 (2018), E2156–E2164.

**[46]** *Github.* `https://github.com`. accessed 16-07-2018.

**[47]** A. van Goethem, I. Kostitsyna, M. van Kreveld, W. Meulemans, M. Sondag, and J. Wulms. “The painter’s problem: covering a grid with colored connected polygons”. In: *International Symposium on Graph Drawing and Network Visualization.* Springer. 2017, pp. 492–505.

**[48]** J. Görtler, C. Schulz, D. Weiskopf, and O. Deussen. “Bubble Treemaps for Uncertainty Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 719–728.

**[49]** M. Greis, P. E. Agroudy, H. Schuff, T. Machulla, and A. Schmidt. "Decision-Making under Uncertainty: How the Amount of Presented Uncertainty Influences User Behavior". In: *Proceedings of the 9th Nordic Conference on Human-Computer Interaction* (2016), pp. 2–5.

**[50]** *Gridmap resources.* `https://github.com/tue-aga/Gridmap`.

**[51]** T. Gschwandtner, M. Bogl, P. Federico, and S. Miksch. "Visual Encodings of Temporal Uncertainty: A Comparative User Study". In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 539–548.

**[52]** D. Guo, J. Chen, A. M. MacEachren, and K. Liao. "A visualization system for space-time and multivariate patterns (vis-stamp)". In: *IEEE Transactions on Visualization and Computer Graphics* 12.6 (2006), pp. 1461–1474.

**[53]** H. Guo, J. Huang, and D. H. Laidlaw. "Representing uncertainty in graph edges: an evaluation of paired visual variables". In: *IEEE Transactions on Visualization and Computer Graphics* 21.10 (2015), pp. 1173–1186.

**[54]** S. Hahn, J. Trümper, D. Moritz, and J. Döllner. "Visualization of varying hierarchies by stable layout of Voronoi treemaps". In: *Proceedings of the International Conference on Information Visualization Theory and Applications.* 2014, pp. 50–58.

**[55]** S. Hahn. "Comparing the Layout Stability of Treemap Algorithms". In: *Proceedings of the HPI research school on service-oriented systems engineering* 95 (2015), pp. 71–79.

**[56]** S. Hahn, J. Bethge, and J. Döllner. "Relative Direction Change: A Topology-based Metric for Layout Stability in Treemaps". In: *Proceedings of the 8th International Conference of Information Visualization Theory and Applications.* 2017, pp. 88–95.

**[57]** F. M. Harper and J. A. Konstan. "The movielens datasets: History and context". In: *ACM Transactions on Interactive Intelligent Systems* 5.4 (2016), p. 19.

**[58]** J.-H. Haunert and L. Sering. "Drawing Road Networks with Focus Regions". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2555–2562.

**[59]** N. Henry Riche and T. Dwyer. "Untangling Euler Diagrams". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 1090–1099.

**[60]** D. D. Hoffman and W. Richards. "Parts of recognition". In: *Cognition* 18.1 (1984), pp. 65–96.

**[61]** N. S. Holliman, A. Coltekin, S. J. Fernstad, M. D. Simpson, K. J. Wilson, and A. J. Woods. "Visual Entropy and the Visualization of Uncertainty". In: *arXiv preprint arXiv:1907.12879* (2019).

**[62]** J. Hullman. "Why Authors Don't Visualize Uncertainty". In: *IEEE transactions on visualization and computer graphics* 26.1 (2019), pp. 130–139.

**[63]** J. Hullman, X. Qiao, M. Correll, A. Kale, and M. Kay. "In Pursuit of Error: A Survey of Uncertainty Visualization Evaluation". In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2019), pp. 903–913.

**[64]** A. Kale, F. Nguyen, M. Kay, and J. Hullman. "Hypothetical Outcome Plots Help Untrained Observers Judge Trends in Ambiguous Data". In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (2019), pp. 892–902.

**[65]** M. Kaufmann, M. van Kreveld, and B. Speckmann. "Subdivision Drawings of Hypergraphs". In: *Proceedings of the 16th International Symposium on Graph Drawing*. 2009, pp. 396–407.

**[66]** R. Khlebnikov, B. Kainz, M. Steinberger, and D. Schmalstieg. "Noise-based volume rendering for the visualization of multivariate volumetric data". In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2926–2935.

**[67]** H. Kim, J. Choo, C. K. Reddy, and H. Park. "Doubly supervised embedding based on class labels and intrinsic clusters for high-dimensional data visualization". In: *Neurocomputing* 150 (2015), pp. 570–582.

**[68]** G. Kindlmann and C. Scheidegger. "An algebraic process for visualization design". In: *IEEE transactions on visualization and computer graphics* 20.12 (2014), pp. 2181–2190.

**[69]** C. Kinkeldey, J. Mason, A. Klippel, and J. Schiewe. "Evaluation of noise annotation lines: using noise to represent thematic uncertainty in maps". In: *Cartography and Geographic Information* 41.5 (2014), pp. 430–439.

**[70]** N. Kong, J. Heer, and M. Agrawala. "Perceptual guidelines for creating rectangular treemaps". In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 990–998.

**[71]** I. Kostitsyna, M. Löffler, M. Sondag, W. Sonke, and J. Wulms. "The hardness of Witness puzzles". In: *34th European Workshop on Computational Geometry*. 2018.

**[72]** M. K. Kuhner and J. Yamato. "Practical Performance of Tree Comparison Metrics". In: *Systematic Biology* 64.2 (2015), pp. 205–214.

**[73]** L. J. Latecki and R. Lakämper. “Convexity rule for shape decomposition based on discrete contour evolution”. In: *Computer Vision and Image Understanding* 73.3 (1999), pp. 441–454.

**[74]** X. Liu, Y. Hu, S. North, and H.-W. Shen. “CorrelatedMultiples: Spatially Coherent Small Multiples With Constrained Multi-Dimensional Scaling”. In: *Computer Graphics Forum* 37.1 (2018), pp. 7–18.

**[75]** M. Löffler and W. Meulemans. “Discretized Approaches to Schematization”. In: *Proceedings of the 29th Canadian Conference on Computational Geometry.* 2017, pp. 220–225.

**[76]** A. Longjas, E. F. Legara, and C. Monterola. “Power law mapping in human area perception”. In: *International Journal of Modern Physics C* 22.5 (2011), pp. 495–503.

**[77]** L. Lu, S. Fan, M. Huang, W. Huang, and R. Yang. “Golden Rectangle Treemap”. In: *Journal of Physics: Conference Series* 787.1 (2017).

**[78]** Y. Lu, J.-M. Lien, M. Ghosh, and N. M. Amato. “$\alpha$-decomposition of polygons”. In: *Computers & Graphics* 36.5 (2012), pp. 466–476.

**[79]** A. M. MacEachren, R. E. Roth, J. O’Brien, B. Li, D. Swingley, and M. Gahegan. “Visual semiotics & uncertainty visualization: An empirical study”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2496–2505.

**[80]** J. D. Mackinlay. “Automating the Design of Graphical Presentations of Relational Information”. In: *ACM Transactions on Graphics* 5.2 (1986), pp. 110–141.

**[81]** E. Mäkinen. “How to draw a hypergraph”. In: *International Journal of Computer Mathematics* 34 (1990), pp. 177–185.

**[82]** B. B. Mandelbrot. “How Long Is the Coast of Britain”. In: *The Fractal Geometry of Nature.* W. H. Freeman, New York, 1983, pp. 25–33.

**[83]** G. McNeill and S. A. Hale. “Generating tile maps”. In: *Computer Graphics Forum* 36.3 (2017), pp. 435–445.

**[84]** Meertens Instituut, KNAW. *Nederlandse Voornamenbank.* `https://www.meertens.knaw.nl/nvb`. Accessed on 30-05-2016. 2013.

**[85]** W. Meulemans, N. Henry Riche, B. Speckmann, B. Alper, and T. Dwyer. “KelpFusion: A hybrid set visualization technique”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.11 (2013), pp. 1846–1858.

**[86]** W. Meulemans, J. Dykes, A. Slingsby, C. Turkay, and J. Wood. “Small Multiples with Gaps”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 381–390.

**[87]** W. Meulemans, M. Sondag, and B. Speckmann. “A Simple Pipeline for Coherent Grid Maps”. In: *IEEE Transactions on Visualization and Computer Graphics* (2020), To appear.

**[88]** X. Mi and D. DeCarlo. “Separating parts from 2d shapes using relatability”. In: *IEEE International Conference on Computer Vision.* 2007, pp. 1–8.

**[89]** S. Mittelstädt, A. Stoffel, and D. A. Keim. “Methods for compensating contrast effects in information visualization”. In: *Computer Graphics Forum.* Vol. 33. 3. Wiley Online Library. 2014, pp. 231–240.

**[90]** T. Munzner. *Visualization analysis and design.* CRC press, 2014.

**[91]** H. Nagamochi and Y. Abe. “An approximation algorithm for dissecting a rectangle into rectangles with specified areas”. In: *Discrete Applied Mathematics* 155.4 (2007), pp. 523–537.

**[92]** New York Times. *How the Rulings Affect Gay Couples.* Accessed April 2020. June 2013. URL: `http://www.nytimes.com/interactive/2013/06/26/us/scotus-gay-marriage.html`.

**[93]** Q. H. Nguyen and P. Eades. “Towards faithful graph visualizations”. In: *arXiv preprint arXiv:1701.00921* (2017).

**[94]** N. Papanelopoulos, Y. Avrithis, and S. Kollias. “Revisiting the medial axis for planar shape decomposition”. In: *Computer Vision and Image Understanding* 179 (2019), pp. 66–78.

**[95]** H. Park. *Gay Marriage State by State: From a Few States to the Whole Nation.* Accessed April 2020. Mar. 2015. URL: `http://www.nytimes.com/interactive/2015/03/04/us/gay-marriage-state-by-state.html`.

**[96]** K. Powell, R. Harris, and F. Cage. *How voter-friendly is your state?* Accessed April 2020. Oct. 2014. URL: `http://www.theguardian.com/us-news/ng-interactive/2014/oct/22/-sp-voting-rights-identification-how-friendly-is-your-state`.

**[97]** H. C. Purchase. “Metrics for graph drawing aesthetics”. In: *Journal of Visual Languages & Computing* 13.5 (2002), pp. 501–516.

**[98]** H. C. Purchase, R. F. Cohen, and M. I. James. “An experimental study of the basis for graph drawing algorithms”. In: *Journal of Experimental Algorithmics (JEA)* 2 (1997), 4–es.

**[99]** H. C. Purchase, R. F. Cohen, and M. I. James. “Validating graph drawing aesthetics”. In: *Proceedings of the Symposium on Graph Drawing.* 1996, pp. 435–446.

**[100]** R. Radburn. *Go with the Flow: Commuting & Migration flows within London.* Accessed April 2020. Mar. 2016. URL: `https://public.tableau.com/profile/robradburn/#!/vizhome/ODMpasLondonAftertheFlood/GowiththeFlow`.

**[101]** R. Radburn. *Home Truths in London.* Accessed April 2020. Apr. 2020. URL: `https://public.tableau.com/profile/robradburn/#!/vizhome/HomeTruthsinLondon/LondonTenure`.

**[102]** N. Richards. *Do tile maps need to have regular shapes?* Accessed April 2020. Feb. 2019. URL: `https://questionsindataviz.com/2019/02/02/do-tile-maps-need-to-have-regular-shapes/`.

**[103]** N. Richards. *How do you tile the world?* Accessed April 2020. Nov. 2017. URL: `https://questionsindataviz.com/2017/11/05/how-do-you-tile-the-world/`.

**[104]** N. Richards. *When are two maps better than one?* Accessed April 2020. May 2019. URL: `https://questionsindataviz.com/2019/05/21/when-are-two-maps-better-than-one/`.

**[105]** N. Richards. *Where are the Africa grid/tile maps?* Accessed April 2020. Sept. 2016. URL: `https://questionsindataviz.com/2016/09/01/where-are-the-africa-gridtile-maps/`.

**[106]** D. Sacha, H. Senaratne, B. C. Kwon, G. Ellis, and D. A. Keim. “The Role of Uncertainty, Awareness, and Trust in Visual Analytics”. In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 240–249.

**[107]** W. Scheibel, C. Weyand, and J. Döllner. “EvoCells – A Treemap Layout Algorithm for Evolving Tree Data”. In: *Proceedings of the International Conference on Information Visualization Theory and Applications.* 2018, pp. 273–280.

**[108]** C. Schulz, K. Schatz, M. Krone, M. Braun, T. Ertl, and D. Weiskopf. “Uncertainty Visualization for Secondary Structures of Proteins”. In: *2018 IEEE Pacific Visualization Symposium.* 2018, pp. 96–105.

**[109]** *Scitools.* `https://scitools.com`.

**[110]** T. Shaw. *Good Data Visualization Practice: Tile Grid Maps.* Accessed April 2020. Apr. 2016. URL: `https://www.forumone.com/ideas/good-data-visualization-practice-tile-grid-maps-0/`.

**[111]** J. Shi and C. Tomasi. “Good features to track”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1994, pp. 593–600.

**[112]** B. Shneiderman. “Tree Visualization with Tree-maps: a 2D Space-filling Approach”. In: *ACM Transactions on Graphics* 11.1 (1992), pp. 92–99.

**[113]** B. Shneiderman and M. Wattenberg. “Ordered treemap layouts”. In: *Proceedings of the IEEE Symposium on Information Visualization.* 2001, pp. 73–78.

**[114]** K. Siddiqi and B. B. Kimia. “Parts of visual form: Computational aspects”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.3 (1995), pp. 239–251.

**[115]** P. Simonetto and D. Auber. “Visualise Undrawable Euler Diagrams”. In: *Proceedings of the 12th Conference on Information Visualisation.* 2008, pp. 594–599.

**[116]** P. Simonetto, D. Auber, and D. Archambault. “Fully Automatic Visualisation of Overlapping Sets”. In: *Computer Graphics Forum* 28.3 (2009), pp. 967–974.

**[117]** M. Singh and D. D. Hoffman. “Part-based representations of visual shape and implications for visual cognition”. In: *Advances in psychology* 130 (2001), pp. 401–459.

**[118]** M. Singh, G. D. Seyranian, and D. D. Hoffman. “Parsing silhouettes: The short-cut rule”. In: *Perception & Psychophysics* 61.4 (1999), pp. 636–660.

**[119]** A. Slingsby. “Tilemaps for summarising multivariate geographical variation”. In: *Proceedings of the Workshop on Visual Summarization and Report Generation.* 2018.

**[120]** A. Slingsby, J. Dykes, and J. Wood. “Rectangular Hierarchical Cartograms for Socio-Economic Data”. In: *Journal of Maps* 6.1 (2010), pp. 330–345.

**[121]** A. Slingsby, J. Dykes, and J. Wood. “Exploring Uncertainty in Geodemographics with Interactive Graphics”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2545–2554.

**[122]** A. Slingsby, M. Kelly, and J. Dykes. “OD maps for showing changes in Irish female migration between 1851 and 1911”. In: *Environment and Planning A* 46.12 (2014), pp. 2795–2797.

**[123]** K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. “Towards Objective Measures of Algorithm Performance Across Instance Space”. In: *Computers & Operations Research* 45 (2014), pp. 12–24.

**[124]** M. Sondag, W. Meulemans, S. Nickel, M. Nollenburg, M. Chimani, and J. Peltonen. "Computing Stable Demers Cartograms". In: *International Symposium on Graph Drawing and Network Visualization.* 2019.

**[125]** M. Sondag, W. Meulemans, C. Schulz, K. Verbeek, D. Weiskopf, and B. Speckmann. "Uncertainty Treemaps". In: *2020 IEEE Pacific Visualization Symposium.* 2020, pp. 111–120.

**[126]** M. Sondag, B. Speckmann, and K. Verbeek. "Stable Treemaps via Local Moves". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 729–738.

**[127]** S. S. Stevens. "On the psychophysical law". In: *Psychological Review* 64.3 (1957), pp. 153–181.

**[128]** S. S. Stevens. "The psychophysics of sensory function". In: *American Scientist* 48.2 (1960), pp. 226–253.

**[129]** T. Sugibuchi, N. Spyratos, and E. Siminenko. "A framework to analyze information visualization based on the functional data model". In: *13th International Conference Information Visualisation.* IEEE. 2009, pp. 18–24.

**[130]** E. Sullivan, M. Sondag, I. Rutter, W. Meulemans, S. Cunningham, B. Speckmann, and M. Alfano. "Can real social epistemic networks deliver the wisdom of crowds?" In: *Oxford Studies in Experimental Philosophy.* Vol. 3. Oxford University Press. 2019.

**[131]** E. Sullivan, M. Sondag, I. Rutter, W. Meulemans, S. Cunningham, B. Speckmann, and M. Alfano. "Vulnerability in social epistemic networks". In: *International Journal of Philosophical Studies* (2020), pp. 1–23.

**[132]** R. Szeliski. *Computer Vision: Algorithms and Applications.* Springer, 2010.

**[133]** S. Tak and A. Cockburn. "Enhanced spatial stability with Hilbert and Moore treemaps". In: *IEEE Transactions on Visualization and Computer Graphics* 19.1 (2013), pp. 141–148.

**[134]** M. Teghtsoonian. "The judgment of size". In: *The American Journal of Psychology* 78.3 (1965), pp. 392–402.

**[135]** A. Telea. "Combining Extended Table Lens and Treemap Techniques for Visualizing Tabular Data". In: *Proceedings of the VGTC Conference on Visualization.* 2006, pp. 120–127.

**[136]** *The Movie Database.* `www.themoviedb.org`. accessed 10-02-2018.

**[137]** *Treemap resources.* `https://eduardovernier.github.io/dynamic-treemap-resources-eurovis`.

**[138]** A. Tribou and K. Collins. *This Is How Fast America Changes Its Mind.* Accessed April 2020. June 2015. URL: `http://www.bloomberg.com/graphics/2015-pace-of-social-change/`.

**[139]** Y. Tu and H.-W. Shen. “Visualizing changes of hierarchical data using treemaps”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1286–1293.

**[140]** E. R. Tufte. *The visual display of quantitative information.* Vol. 2. Graphics press Cheshire, CT, 2001.

**[141]** W. Tutte. *Graph Theory.* Addison-Wesley, Menlo Park (CA), USA, 1984.

**[142]** T. Tuytelaars and K. Mikolajczyk. “Local Invariant Feature Detectors: A Survey”. In: *Foundations and Trends in Computer Graphics and Vision* 3.3 (2007), pp. 177–280.

**[143]** *Uncertainty Treemaps.* `https://github.com/tue-aga/UncertaintyTreemaps`.

**[144]** United Nations. *UN Comtrade database.* `https://comtrade.un.org`. Accessed on 15-02-2017.

**[145]** *USGS Earthquakes.* `https://earthquake.usgs.gov/earthquakes/browse/stats.php`. accessed 03-07-2018.

**[146]** P. M. Vaidya. “Geometry helps in matching”. In: *SIAM Journal on Computing* 18.6 (1989), pp. 1201–1225.

**[147]** E. Vernier, J. Comba, and A. Telea. “A Stable Greedy Insertion Treemap Algorithm for Software Evolution Visualization”. In: *IEEE Conference on Graphics, Patterns and Images.* 2018, pp. 158–165.

**[148]** E. Vernier, J. Comba, and A. Telea. “Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization”. In: *IEEE Conference on Software Visualization.* 2018, pp. 96–106.

**[149]** E. Vernier, M. Sondag, J. Comba, B. Speckmann, A. Telea, and K. Verbeek. “Quantitative Comparison of Time-Dependent Treemaps”. In: *Computer Graphics Forum* (2020).

**[150]** R. Vliegen, J. J. van Wijk, and E. J. van der Linden. “Visualizing Business Data with Generalized Treemaps”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 789–796.

**[151]** T. Walsh. “Hypermaps versus bipartite maps”. In: *Journal of Combinatorial Theory* 18 (1975), pp. 155–163.

**[152]** M. Wattenberg. "A note on space-filling visualizations and space-filling curves". In: *Proceedings of the IEEE Symposium on Information Visualization.* 2005.

**[153]** K. Wongsuphasawat. *A semi-automatic way to create your own grid map.* Accessed April 2020. Jan. 2016. URL: `https://medium.com/%5C@kristw/creating-grid-map-for-thailand-397b53a4ecf`.

**[154]** K. Wongsuphasawat. *Whose Grid Map is better? Quality Metrics for Grid Map Layouts.* Accessed April 2020. Jan. 2016. URL: `https://medium.com/%5C@kristw/whose-grid-map-is-better-quality-metrics-for-grid-map-layouts-e3d6075d9e80`.

**[155]** J. Wood, D. Badawood, J. Dykes, and A. Slingsby. "BallotMaps: Detecting name bias in alphabetically ordered ballot papers". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2384–2391.

**[156]** J. Wood and J. Dykes. "Spatially ordered treemaps". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1348–1355.

**[157]** J. Wood, A. Slingsby, and J. Dykes. "Visualizing the dynamics of London's bicycle-hire scheme". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 46.4 (2011), pp. 239–251.

**[158]** *Worldbank indicators.* `https://data.worldbank.org/indicator/`. accessed 04-07-2018.

**[159]** *Worldbank infant death indicator.* `https://data.worldbank.org/indicator/SH.DTH.IMRT`. accessed 04-07-2018.

**[160]** E. F. Young, C. C. Chu, and Z. C. Shen. "Twin binary sequences: A nonredundant representation for general nonslicing floorplan". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22.4 (2003), pp. 457–469.

# Summary

## Algorithms for Coherent Rectangular Visualizations

To gain insight into the large amounts of data that chronicle our life, work and world, we often turn to visualizations to use our powerful perceptual system for (parts of) the analysis. For a visualization to be effective, the visualization has to visually represent all properties of the data that the viewer requires for the task they want to perform. Moreover, an effective visualization has to be coherent: relations between data items are visually represented, and visible relations are present in the data. A coherent visualization thus prevents false patterns from emerging in the visualization, and preserves patterns that exist in the data. Our focus in this thesis lies on rectangular visualizations, that is, visualizations that use rectangles as their fundamental building blocks. We describe several new algorithms that transform data into coherent rectangular visualizations for four different data types.

We first study temporal coherence for treemaps, which visualize hierarchical data using nested rectangles whose areas match the data values. When the data varies over time, we can generate a treemap for each time step. The quality of a treemapping algorithm is then determined by two factors: (i) the visual quality, which indicates how well we can determine the values of the data for a single treemap and (ii) the temporal coherence or stability, which indicates how coherent the changes in the data are with the changes in the treemap. Ideally, small changes in the data should result in small changes in the treemap. We propose a novel stable treemapping algorithm that has high visual quality. Whereas existing treemapping algorithms generally recompute the treemap every time the input changes, our algorithm changes the layout of the treemap using only local modifications. This approach gives us direct control over the stability, and in contrast to existing treemapping algorithms, can also generate non-sliceable treemap layouts. We prove that using these non-sliceable layouts can result in treemaps of higher visual quality. To verify the efficacy of the new algorithm as well as existing treemapping algorithms, we perform an extensive quantitative evaluation of rectangular treemapping algorithms for time-varying hierarchical data. To this end, we first propose a new method to measure the stability of time-varying treemaps which explicitly considers the input data. Additionally, we identify four representative features of datasets that influence the performance of treemapping algorithms. We use these features to propose a novel classification scheme for time-varying hierarchical datasets. We experimentally test the validity of this classification on a large number of datasets, and use this classification to compare and evaluate treemapping algorithms on a variety of datasets.

Second, we study the coherence between data and uncertainty when visualizing uncertain hierarchical data using treemaps. To visualize uncertainty in a treemap, we identify two key, but conflicting, requirements: (i) to easily assess the data value of a node in the hierarchy, the area of its rectangle should directly match its data value, and (ii) to ensure coherence and facilitate comparison between data and uncertainty, uncertainty must be encoded using the same visual variable as the data, that is, area. We present Uncertainty Treemaps which meet both requirements simultaneously using the novel concept of hierarchical uncertainty masks. We define a new cost function that measures the quality of Uncertainty Treemaps and show how to adapt existing treemapping algorithms to support uncertainty masks. Finally, we demonstrate the quality of our technique through a computational experiment on real-world datasets.

Third, we improve upon the spatial coherence of grid maps: spatial arrangements of simple tiles (often squares or hexagons) each of which represents a spatial element. An effective grid map is coherent with the underlying spatial dimension: the tiles maintain properties such as contiguity, neighborhoods and identifiability of the corresponding spatial elements, while the grid map as a whole maintains the global shape of the input. Of particular importance are salient local features of the global shape which need to be represented by tiles assigned to the appropriate spatial elements. State-of-the-art techniques can adequately deal only with simple cases, such as a close-to-uniform spatial dimension or global shapes that have few characteristic features. We introduce a simple fully-automated 3-step pipeline for computing high-quality coherent grid maps. Each step relies on well-established solutions: shape decomposition based on salient features, tile-based Mosaic Cartograms, and point-set matching. We provide an implementation, demonstrate the efficacy of our approach on various complex datasets and compare it to the state-of-the-art.

Finally, we propose a new way to visualize spatial set data. We assume that the nodes of the sets are positioned on a grid. Each node is represented by a cell in the grid and each set is represented by a single connected polygon overlapping exactly those cells that correspond to the nodes in the set. For the case where there are two sets, we derive a necessary and sufficient condition to efficiently recognize whether we can represent each set with a single connected polygon. Additionally, we show that the visual complexity of the polygon in each cell can be bounded by a small constant.

# Curriculum Vitae

Max Sondag was born on the 5th of March, 1993, in Uden, the Netherlands. He finished secondary education at Udens College in Uden, the Netherlands in 2011. He then studied Computer Science and Engineering at TU Eindhoven in Eindhoven, obtaining his Bachelor's degree in 2014. He obtained his Master's degree in Computer Science and Engineering (cum laude) in 2016 at the same university. Since 2016 he has been a PhD student under the supervision of Bettina Speckmann, and co-supervised by Wouter Meulemans. The main results of his research as a PhD student are presented in this dissertation.

## Titles in the IPA Dissertation Series since 2017

**M.J. Steindorfer**. *Efficient Immutable Collections.* Faculty of Science, UvA. 2017-01

**W. Ahmad**. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

**D. Guck**. *Reliable Systems – Fault tree analysis via Markov reward automata.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

**H.L. Salunkhe**. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors.* Faculty of Mathematics and Computer Science, TU/e. 2017-04

**A. Krasnova**. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT).* Faculty of Science, Mathematics and Computer Science, RU. 2017-05

**A.D. Mehrabi**. *Data Structures for Analyzing Geometric Data.* Faculty of Mathematics and Computer Science, TU/e. 2017-06

**D. Landman**. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities.* Faculty of Science, UvA. 2017-07

**W. Lueks**. *Security and Privacy via Cryptography – Having your cake and eating it too.* Faculty of Science, Mathematics and Computer Science, RU. 2017-08

**A.M. Şutîi**. *Modularity and Reuse of Domain-Specific Languages: an exploration with MetaMod.* Faculty of Mathematics and Computer Science, TU/e. 2017-09

**U. Tikhonova**. *Engineering the Dynamic Semantics of Domain Specific Languages.* Faculty of Mathematics and Computer Science, TU/e. 2017-10

**Q.W. Bouts**. *Geographic Graph Construction and Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2017-11

**A. Amighi**. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

**S. Darabi**. *Verification of Program Parallelization.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

**J.R. Salamanca Tellez**. *Coequations and Eilenberg-type Correspondences*. Faculty of Science, Mathematics and Computer Science, RU. 2018-03

**P. Fiterău-Broştean**. *Active Model Learning for the Analysis of Network Protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2018-04

**D. Zhang**. *From Concurrent State Machines to Reliable Multi-threaded Java Code*. Faculty of Mathematics and Computer Science, TU/e. 2018-05

**H. Basold**. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic*. Faculty of Science, Mathematics and Computer Science, RU. 2018-06

**A. Lele**. *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems*. Faculty of Mathematics and Computer Science, TU/e. 2018-07

**N. Bezirgiannis**. *Abstract Behavioral Specification: unifying modeling and programming*. Faculty of Mathematics and Natural Sciences, UL. 2018-08

**M.P. Konzack**. *Trajectory Analysis: Bridging Algorithms and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2018-09

**E.J.J. Ruijters**. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

**F. Yang**. *A Theory of Executability: with a Focus on the Expressivity of Process Calculi*. Faculty of Mathematics and Computer Science, TU/e. 2018-11

**L. Swartjes**. *Model-based design of baggage handling systems*. Faculty of Mechanical Engineering, TU/e. 2018-12

**T.A.E. Ophelders**. *Continuous Similarity Measures for Curves and Surfaces*. Faculty of Mathematics and Computer Science, TU/e. 2018-13

**M. Talebi**. *Scalable Performance Analysis of Wireless Sensor Network*. Faculty of Mathematics and Computer Science, TU/e. 2018-14

**R. Kumar**. *Truth or Dare: Quantitative security analysis using attack trees*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

**M.M. Beller**. *An Empirical Evaluation of Feedback-Driven Software Development*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

**M. Mehr**. *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems*. Faculty of

Mathematics and Computer Science, TU/e. 2018-17

**M. Alizadeh**. *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations.* Faculty of Mathematics and Computer Science, TU/e. 2018-18

**P.A. Inostroza Valdera**. *Structuring Languages as Object-Oriented Libraries.* Faculty of Science, UvA. 2018-19

**M. Gerhold**. *Choice and Chance - Model-Based Testing of Stochastic Behaviour.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20

**A. Serrano Mena**. *Type Error Customization for Embedded Domain-Specific Languages.* Faculty of Science, UU. 2018-21

**S.M.J. de Putter**. *Verification of Concurrent Systems in a Model-Driven Engineering Workflow.* Faculty of Mathematics and Computer Science, TU/e. 2019-01

**S.M. Thaler**. *Automation for Information Security using Machine Learning.* Faculty of Mathematics and Computer Science, TU/e. 2019-02

**Ö. Babur**. *Model Analytics and Management.* Faculty of Mathematics and Computer Science, TU/e. 2019-03

**A. Afroozeh and A. Izmaylova**. *Practical General Top-down Parsers.* Faculty of Science, UvA. 2019-04

**S. Kisfaludi-Bak**. *ETH-Tight Algorithms for Geometric Network Problems.* Faculty of Mathematics and Computer Science, TU/e. 2019-05

**J. Moerman**. *Nominal Techniques and Black Box Testing for Automata Learning.* Faculty of Science, Mathematics and Computer Science, RU. 2019-06

**V. Bloemen**. *Strong Connectivity and Shortest Paths for Checking Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-07

**T.H.A. Castermans**. *Algorithms for Visualization in Digital Humanities.* Faculty of Mathematics and Computer Science, TU/e. 2019-08

**W.M. Sonke**. *Algorithms for River Network Analysis.* Faculty of Mathematics and Computer Science, TU/e. 2019-09

**J.J.G. Meijer**. *Efficient Learning and Analysis of System Behavior.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-10

**P.R. Griffioen**. *A Unit-Aware Matrix Language and its Application in Control and Auditing.* Faculty of Science, UvA. 2019-11

**A.A. Sawant**. *The impact of API evolution on API consumers and how this can be affected by API producers and language designers.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2019-12

**W.H.M. Oortwijn**. *Deductive Techniques for Model-Based Concurrency Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-13

**M.A. Cano Grijalba**. *Session-Based Concurrency: Between Operational and Declarative Views.* Faculty of Science and Engineering, RUG. 2020-01

**T.C. Nägele**. *CoHLA: Rapid Cosimulation Construction.* Faculty of Science, Mathematics and Computer Science, RU. 2020-02

**R.A. van Rozen**. *Languages of Games and Play: Automating Game Design & Enabling Live Programming.* Faculty of Science, UvA. 2020-03

**B. Changizi**. *Constraint-Based Analysis of Business Process Models.* Faculty of Mathematics and Natural Sciences, UL. 2020-04

**N. Naus**. *Assisting End Users in Workflow Systems.* Faculty of Science, UU. 2020-05

**J.J.H.M. Wulms**. *Stability of Geometric Algorithms.* Faculty of Mathematics and Computer Science, TU/e. 2020-06

**T.S. Neele**. *Reductions for Parity Games and Model Checking.* Faculty of Mathematics and Computer Science, TU/e. 2020-07

**P. van den Bos**. *Coverage and Games in Model-Based Testing.* Faculty of Science, RU. 2020-08

**M.F.M. Sondag**. *Algorithms for Coherent Rectangular Visualizations.* Faculty of Mathematics and Computer Science, TU/e. 2020-09