VISUALIZATION OF DYNAMIC MULTIDIMENSIONAL AND HIERARCHICAL DATASETS

EDUARDO FACCIN VERNIER

Cover: Trails generated by the C-UMAP method projecting the *carto-lastd* dataset, as seen in Chapter 6.

Visualization of Dynamic Multidimensional and Hierarchical Datasets

Eduardo Faccin Vernier
PhD Thesis

This thesis is the result of a joint PhD between the Federal University of Rio Grande do Sul and the University of Groningen.

university of groningen

Visualization of Dynamic Multidimensional and Hierarchical Datasets

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans, and
to obtain the degree of PhD at the
University of Rio Grande do Sul
on the authority of
Rector Magnificus Prof. C. Bulhões Mendes.

Double PhD Degree

This thesis will be defended in public on

Weekday DD Month YYYY at HH.MM hours

by

Eduardo Faccin Vernier

born on March 30th, 1995
in Porto Alegre, Brazil

Supervisors
Prof. dr. Alexandru Cristian Telea
Prof. dr. João Luiz Dihl Comba

Assessment committee
Prof. Assessment prof1
Prof. Assessment prof2
Prof. Assessment prof3
Prof. Assessment prof4

Just about anything looks better from a distance.
— Haruki Murakami

ABSTRACT

When it comes to tools and techniques designed to help understanding complex abstract data, visualization methods play a prominent role. They enable human operators to leverage their pattern finding, outlier detection, and questioning abilities to visually reason about a given dataset. Many methods exist that create suitable and useful visual representations of *static* abstract, non-spatial, data. However, for *temporal* abstract, non-spatial, datasets, in which the data changes and evolves through time, far fewer visualization techniques exist.

This thesis focuses on the particular cases of temporal hierarchical data representation via dynamic treemaps, and temporal high-dimensional data visualization via dynamic projections. We tackle the joint question of how to extend projections and treemaps to stably, accurately, and scalably handle temporal multivariate and hierarchical data. The literature for static visualization techniques is rich and the state-of-the-art methods have proven to be valuable tools in data analysis. Their temporal/dynamic counterparts, however, are not as well studied, and, until recently, there were few hierarchical and high-dimensional methods that explicitly took into consideration the temporal aspect of the data. In addition, there are few or no metrics to assess the quality of these temporal mappings, and even fewer comprehensive benchmarks to compare these methods.

This thesis addresses the abovementioned shortcomings. For both dynamic treemaps and dynamic projections, we propose ways to accurately measure temporal stability; we evaluate existing methods considering the tradeoff between stability and visual quality; and we propose new methods that strike a better balance between stability and visual quality than existing state-of-the-art techniques. We demonstrate our methods with a wide range of real-world data, including an application of our new dynamic projection methods to support the analysis and classification of hyperkinetic movement disorder data.

SAMENVATTING

Visualisatiemethoden spelen een hoofdrol in de context van gereedschap en technieken voor het begrijpen van complexe en abstracte gegevens. Deze methoden stellen gebruikers in staat om hun patroonherkenning-, uitschieterdetectie-, en vraagstellingsvermogens te gebruiken om visueel te kunnen redeneren over een gegeven dataverzameling. Veel methodes bestaan om toepasselijke en nuttige visuele representaties te creëren vanuit *statische*, niet-ruimtelijke, gegevens. Voor *tijdsafhankelijke*, abstracte, en niet-ruimtelijke gegevens, waarin de data verandert en evolueert over de tijd, veel minder visualisatietechnieken zijn beschikbaar.

Dit proefschrift behandelt het specifieke geval van tijdsafhankelijke hiërarchische data-afbeelding via dynamische *treemaps* en van tijdsafhankelijke hoogdimensionale datavisualisatie via projecties. We benaderen de gezamenlijke vraag van hoe men projecties en *treemaps* kan uitbreiden om tijdsafhankelijke en hoogdimensionale gegevens weer te geven op een stabiele, accurate en schaalbare manier. De literatuur voor statische visualisatietechnieken is rijk en *state-of-the-art* methoden hebben zich bewezen als waardevolle instrumenten voor data-analyse. De varianten van deze methodes voor tijdsafhankelijke (dynamische) data zijn echter niet goed bestudeerd; tot kort waren er maar weining hiërarchische en hoogdimensionale methodes die het tijdsafhankelijke aspect van de data expliciet beschouwden. Bovendien zijn er weinig of soms geen metrieken geschikt om de kwaliteit van deze tijdsafhankelijke methoden waar te nemen, en nog minder uitgebreide *benchmarks* voor het vergelijken van dergelijke methodes.

Dit proefschrift benadert en beantwoordt de bovengenoemde beperkingen. Voor beide dynamische *treemaps* en projecties presenteren wij manieren om hun tijdsafhankelijke stabiliteit nauwkeurig te meten; wij evalueren bestaande methodes in het licht van het balans tussen stabiliteit en visuele kwaliteit; en wij presenteren nieuwe methodes die een beter balans geven tussen stabiliteit en visuele kwaliteit dan bestaande top-kwaliteit methodes. We illustreren onze methodes met een brede collectie van reële gegevens en ook een toepassing van onze nieuwe dynamische projectiemethodes voor de analyse en classificatie van hyperkinetische bewegingsstoornisdata.

RESUMO

Quando se trata de ferramentas e técnicas projetadas para ajudar na compreensão dados abstratos complexos, métodos de visualização desempenham um papel proeminente. Eles permitem que os operadores humanos alavanquem suas habilidades de descoberta de padrões, detecção de valores discrepantes, e questionamento visual para a raciocinar sobre um determinado conjunto de dados. Existem muitos métodos que criam representações visuais adequadas e úteis de para dados *estáticos*, abstratos, e não-espaciais. No entanto, para dados *temporais*, abstratos, e não-espaciais, isto é, dados que mudam e evoluem no tempo, existem poucas técnicas apropriadas.

Esta tese concentra-se nos casos específicos de representação temporal de dados hierárquicos por meio de treemaps dinâmicos, e visualização temporal de dados de alta dimensionalidade via projeções dinâmicas. Nós abordar a questão conjunta de como estender projeções e treemaps de forma estável, precisa e escalável para lidar com conjuntos de dados hierárquico-temporais e multivariado-temporais. Em ambos os casos, a literatura para técnicas estáticas é rica e os métodos estado da arte provam ser ferramentas valiosas em análise de dados. Suas contrapartes temporais/dinâmicas, no entanto, não são tão bem estudadas e, até recentemente, existiam poucos métodos hierárquicos e de alta dimensão que explicitamente levavam em consideração o aspecto temporal dos dados. Além disso, existiam poucas métricas para avaliar a qualidade desses mapeamentos visuais temporais, e ainda menos benchmarks abrangentes para comparação esses métodos.

Esta tese aborda as deficiências acima mencionadas para treemaps dinâmicos e projeções dinâmicas. Propomos maneiras de medir com precisão a estabilidade temporal; avaliamos os métodos existentes, considerando o compromisso entre estabilidade e qualidade visual; e propomos novos métodos que atinjem um melhor equilíbrio entre estabilidade e a qualidade visual do que as técnicas estado da arte atuais. Demonstramos nossos métodos com uma ampla gama de dados do mundo real, incluindo uma aplicação de nossos novos métodos de projeção dinâmica para apoiar a análise e classificação dos dados de transtorno de movimentos.

PUBLICATIONS

This thesis is the result of the following publications:

- Quantitative Comparison of Treemap Techniques for Time-Dependent Hierarchies (poster) (Vernier et al., 2017) [1]
- Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization (Vernier et al., 2018b) [2]
- A Stable Greedy Insertion Treemap Algorithm for Software Evolution Visualization (Vernier et al., 2018a)
- Selecting and Sharing Multidimensional Projection Algorithms (Espadoto et al., 2020a)
- Quantitative Comparison of Time-Dependent Treemaps (Vernier et al., 2020b)
- Quantitative Evaluation of Time-Dependent Multidimensional Projection Techniques (Vernier et al., 2020a)
- Guided Stable Dynamic Projections (Vernier et al., 2021)

1 Best Poster Award at Eurovis 2017
2 Distinguished paper award at VISSOFT 2018

CONTENTS

1 INTRODUCTION

As science and technology evolve, a myriad of data collections is produced. The common denominator of all these data collections is the same – they are regarded as containing useful and usable information from which actionable evidence can be extracted, the latter leading to improvements in many directions – increased sales, better customer support, more accurate assessments of phenomena, or, at a higher level, the advance of understanding of these phenomena that have created the data, and, thereby, an increase of knowledge and advancement to science. These datasets challenge their consumers in many ways. Several relevant aspects that induce such challenges include the *size* of datasets (the 'big data' revolution has made the efficient processing of terrabyte dataset collections a mandatory requirement for most application areas); the *quality* of data (principled statistical analysis and modeling of phenomena captured by data require accurately measured and collected datasets); and the *provenance* of data (one needs to know how the entire end-to-end pipeline of obtaining a given dataset looks like before being able to make strong statements concerning insights derived from the respective data).

Yet, the challenges of understanding data are not limited to the above issues. A separate, and equally important, one regards the *structure* of data itself. By structure, we mean here the relations that connect the measured data samples, also called observations or data points, so as to allow scientists to infer high-level considerations about the underlying phenomena. Data structure regards many aspects, including, but not limited to, the *nature* of the data samples (*e.g.*, these can be quantitative, categorical, text, relational, or multimedia types); the *dimensionality* of the samples (how many independent attributes are measured to yield a single data sample); the *temporality* of data (are the samples part of a single measurement of a phenomenon done at a single moment in time, or are they spread over a time range); and the *organization* of the samples (do the samples create a flat, unstructured, set, or are they organized in a more refined manner, to denote a part-whole relationship).

Among many methods for data analysis, including statistics and machine learning, *visualization* has a special place. Data visualization essentially leverages the skills of the human operator in *e.g.* pattern finding, outlier detection, and at a higher level discovering unknown relationships between the data samples and thereby the ability to pose valuable questions by depicting the measured data via visual representations. Further on, data visualization can be enhanced by *visual analytics*, which leverages the interactive dimension to allow human operators to

view data from various angles, pose questions, and most importantly, formulate, check, and (in)validate hypotheses that ultimately lead to understanding the phenomena that have generated the data in the first place.

Data visualization – and by extension, visual analytics – can be roughly split into two main subfields. *Scientific visualization* is traditionally concerned with the visual exploration of datasets consisting of samples having a relatively low number of dimensions (measurements). More importantly, these data points sample *continuous* phenomena that cover the evolution of natural systems in the 2D or 3D space that describes our physical world, *e.g.*, the flow of fluids, evolution of weather systems, the observable Universe, or chemical, anatomical, or medical phenomena embedded in their respective sciences. While far from having solved all problems in these fields, scientific visualization can exploit the 2D or 3D embedding of the measured samples, and, more importantly, exploit the continuous nature of the sampled phenomena, by using well-understood sampling, aggregation, statistical, and signal reconstruction methodology.

Information visualization aims at leveraging the visual exploration power for all datasets which do not fall within the scientific visualization realm, meaning datasets where samples (a) do not reside in physical 2D or 3D space and/or (b) where the sampled dimensions are not necessarily continuous. An enormous realm of datasets, arguably larger than the scientific visualization ones, falls into this class. Examples include arbitrary data tables in any database or graphs and networks. Major challenges of information visualization stem precisely from the *abstract* nature of the data it needs to depict: Since this data does not (usually) have a physical counterpart, it is (a) far less clear than for the scientific visualization case how to *map* this data to suitable visual representations. On the other hand, by their very definitions, visualizations are perceived as continuous phenomena by users looking at them – whether this regards the *spatial* distribution of visual shapes drawn or the *temporal* dynamics of how visualizations change over time as the depicted data changes. Since abstract data does not have an inherent continuity, the fundamental question arises on how to map non-continuous data to continuous visual representations for a good interpretation. Additional issues concerning big data, such as aggregation and sampling, only complicate the picture.

In this thesis, we attack a part of the above problem by focusing on abstract data that has three specific aspects:

- *Hierarchical:* We consider data that can be organized into a hierarchy. Examples include files and folders stored on a hard disk, organigrams of people in an organization, or, more generally, any data that allows successive grouping and binning via multiple criteria;

- *High-dimensional:* We consider data whose samples consist of a large number (tens or more) of independently measured variables, such as patient records, images treated by deep learning, or any data table having a large number of data columns;
- *Time-dependent:* We consider data that changes over time, also called *temporal* or *dynamic* data. Here, the samples of a dataset consist of both measurements taken at the same time instant and measurements taken at different, consecutive, time instants.

In isolation, information visualization provides many methods to depict data which is hierarchical, high-dimensional, or temporal. However, the combination of at least two of the above three data aspects immediately makes the visualization problem far harder to address. In particular, we are interested in exploring how to visualize abstract datasets which are *hierarchical and time-dependent*, respectively *high-dimensional and time-dependent*. We outline the two separate challenges – visualizing abstract data which is hierarchical and temporal, respectively high-dimensional and temporal, in Secs. 1.1 and 1.2 next.

1.1 VISUALIZING TEMPORAL HIERARCHICAL DATA

Hierarchical datasets are trees, or hierarchies, with weighted nodes. Typically, weights are given for leaf nodes and computed for non-leaf nodes as the sum of their children's weights. One example of a hierarchical dataset is a computer's file system – the "shape of the tree" is given by the organization of the directories, and the weight of the leaf nodes can be defined as a file attribute such as size. Many other examples exist. As already hinted at earlier in this chapter, virtually any tabular dataset can be reduced to such a hierarchy, by grouping rows (samples) successively on multiple criteria, each of them based on the similarity of the rows according to a given table column. Hence, any tabular dataset can lead to multiple hierarchies. Apart from that, inherently hierarchical datasets do exist, such as the file system example mentioned earlier.

The temporal aspect is introduced when data changes over time. Building upon the file system example, if we collect "snapshots", as in a Git repository or periodic backups of a hard drive, our data would be an evolving tree – its organization (hierarchy) changes as we create and delete folders and files, and the weight of the leaf nodes change as files are edited.

There is a range of visual encodings suitable/designed for displaying hierarchical data, including node-link diagrams, icicle plots, sunburst plots, and, the focus of our research, *treemaps*. Given an input weighted tree, treemaps recursively partition a 2D spatial region into cells whose area (and possibly color, shading, and labels) encode the tree's data attributes. Treemaps have several major advantages compared to other visualizations of hierarchical data: They are visually compact – every

single screen pixel is used to convey data, which favors them for visualizing large hierarchies; and they are simple to understand.

Connecting back to our initial example, the first treemaping algorithm (Shneiderman, 1992) was, in fact, designed for the purpose of displaying the contents and use of a file system by Ben Shneiderman:

> *"During 1990, in response to the common problem of a filled hard disk, I became obsessed with the idea of producing a compact visualization of directory tree structures. Since the 80 Megabyte hard disk in the HCIL was shared by 14 users it was difficult to determine how and where space was used. Finding large files that could be deleted, or even determining which users consumed the largest shares of disk space were difficult tasks".*

Compared to other hierarchical visualization methods, treemaps scale well, making use of all available screen pixels to show data and thus able to handle trees of thousands of nodes. As Shneiderman realized when trying to reason and make decisions about the use of space in his lab's hard drives: *"Tree structured node-link diagrams grew too large to be useful, so I explored ways to show a tree in a space-constrained layout."*

Over the years, a range of different treemapping techniques was proposed, each designed to optimize different goals such as cell aspect ratio of cells (tied to readability), order preservation, similarity-based placement, and portrayal of uncertainty. The most common form of treemaps are rectangular treemaps, but a range of alternative models exist, such as Voronoi treemaps (Balzer and Deussen, 2005; Balzer et al., 2005), orthoconvex (Berg et al., 2014), Bubble (Görtler et al., 2018), and Jigsaw treemaps (Wattenberg, 2005).

Yet, not much work has been done regarding dynamic treemaps, that is, treemap methods designed to portray temporal hierarchical datasets ensuring that temporal coherence is kept. In other words: While treemaps work well for depicting any (large) hierarchical dataset, when one adds the requirement that the data is changing, we do not know whether, and how, to use or adapt treemaps to display such data. As hinted at the beginning of this chapter, a major issue here is the *continuity* aspect: As data changes, so will treemaps that depict it. However, how to ensure that what a user sees in terms of visualization changes (in the treemaps) faithfully conveys the change in the underlying data? This is a major requirement for using treemaps for dynamic data. Indeed: If a dataset changed a lot, but the corresponding dynamic treemap visualization would not, then the user would get the false feeling that the data is not changing when it actually does. Conversely, if a dataset changed little over time, but the corresponding dynamic treemap visualization exhibited major changes between its different snapshots (corresponding to different measurement moments of the hierarchical data),

the user would get the false feeling that data is changing a lot when it actually does not. In both above examples, we would have clear cases of *false negatives*, respectively *false positives* (related to interpreting change in the visualization), which are both detrimental for the added value of a visualization. While the above problems are known and recognized, there are no clear solutions to designing treemap algorithms to address them for dynamic data. This is our first key research question (detailed next in this chapter).

1.2 VISUALIZING TEMPORAL MULTIDIMENSIONAL DATA

Multidimensional (or high-dimensional) datasets have a number of observations (also called points or samples) where each observation has many attributes, also called variables, dimensions, or measurements. For datasets with relatively small numbers of observations and dimensions, techniques such as glyphs, parallel coordinate plots, table lenses, and scatterplot matrices can produce accurate and useful visual encodings. If a dataset has a large number of dimensions, roughly more than 4 to 5, however, multidimensional *projections* tend to be the only scalable approach.

Multidimensional projections take data in a high-dimensional space and project it into a lower-dimensional space, usually creating a 2D or 3D scatter plot, which we can directly visualize and reason about. In this transformation, the projection method attempts to create visual patterns that reflect the similarities or structure found in the high-dimensional space. That is, points which are similar – according to any suitable similarity metric – in the high-dimensional space are placed close in the 2D or 3D projection, and conversely.

Reflecting the similarities or structure found in the high-dimensional space can be interpreted in many ways, and the search for the "best" projection method has led to the proposal of a huge number of projection techniques. There are many desirable traits projection methods can have, such as creating high distance or neighborhood preservation maps, scalability, simplicity, interpretability, out-of-sample capability, stability, and ease of use, among others. Optimizing a single one of these traits is already a challenging task that requires tradeoffs regarding the remaining traits. No single current method optimally satisfies all desirable requirements.

The trait that concerns us most in this thesis is *stability*. Stability, or temporal coherence, needs to be taken into account when we project *temporal* multidimensional data, that is, when the multidimensional data changes over time and, as illustrated in the previous section, we have multiple snapshots of the data. Most projection techniques are designed for static data. When used for time-dependent data, they usually fail to create a stable and suitable low-dimensional representation. To follow the analogy with the dynamic treemaps discussed in Sec. 1.1: If

we have a high-dimensional and temporal dataset, current projection methods usually create a visualization in which the observed points either do not change much while their corresponding high-dimensional counterparts change a lot; or they do change a lot whereas the high-dimensional data points only change little. Globally, the problem with projections is the same as that of treemaps – they can produce false negatives and/or false positives which impair the ability of the user to judge about the data dynamics from seeing the visualization of the projected data.

1.3 TEMPORAL COHERENCE

Treemaps and projections are incredibly useful techniques that, due to their compact and easy to interpret design, give unique insights into large and complicated datasets. As mentioned so far, they were initially designed for static datasets. However, given the presence of dynamic datasets, the natural question arises on how to adapt them to handle such data, while avoiding the already discussed false-negative and false-positive problems.

To illustrate the dynamic projections' instability, Fig. 1.1 shows three different methods (G-PCA, TF-PCA (Jolliffe, 1986), and TF-tSNE (van der Maaten and Hinton, 2008)) projecting the same dataset, using a trail-like visual encoding. The *gaussians* dataset is a 100-dimensional dataset of 2000 samples covering 10 distinct isotropic Gaussian distributions that collapse into 10 single points over 10 timesteps. Knowing the dataset, we can tell that G-PCA renders quite faithfully the data dynamics and structure; TF-PCA creates an artificial amount of spiraling; and TF-tSNE creates a very large amount of apparently random and unstable motion that is not present in the data. For the purpose of the illustration in Fig. 1.1, detailed knowledge of the G-PCA, TF-PCA, and TF-tSNE projection methods is not needed. The point being made is that different projection methods show widely different visual insights in the *same* dataset. As such, they clearly cannot be all right – raising the question of which method is the best and, subsequently, how to define what a good method is in this context.

The same effect occurs when we create treemaps for time-dependent datasets. The top row of Fig. 1.2 shows three snapshots/timesteps of the evolution of a simple weighted tree. The next two rows show two different treemapping algorithms creating rectangular treemap representations for the data (NMap and Squarified Treemap). Nmap creates a stable layout; that is, there are no significant changes in the positions of the cells driven by the small changes in the data, and the adjacencies in the layout remain the same over the evolution. In contrast, in the Squarified Treemap layout, cell d (red) keeps changing its relative position. When dealing with more complicated datasets, this movement can happen for multiple cells simultaneously, making it impossible to

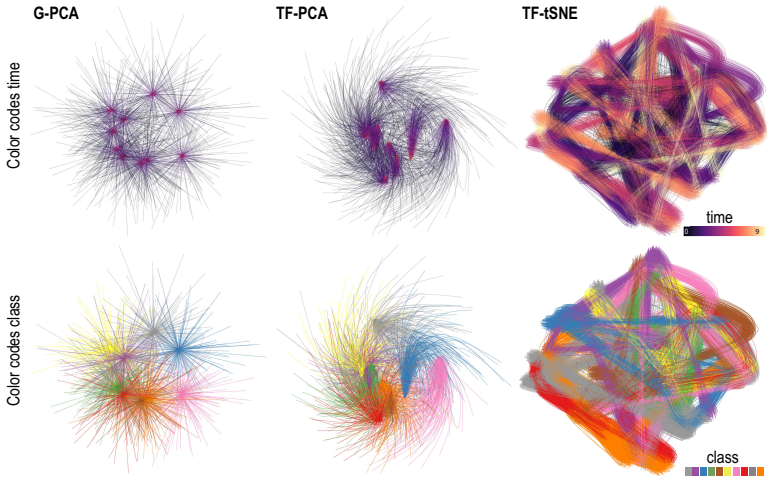


Figure 1.1: A time-dependent collapsing 100-dimensional 10-Gaussian-distributions dataset (2000 points) from Rauber et al. (2016) is visualized by three projection methods. Point trails are colored by time (top) and class (bottom). The images show increasing amounts of instability artefacts.

accurately reason about the data and the change in the data. As for the projection example in Fig. 1.1, the issue is not understanding how NMap or Squarified Treemap work. Rather, the higher-level question is that (at least) one of these methods is suboptimal, and, as a consequence, how to measure the quality of a dynamic treemapping method.

To create faithful and useful representation of temporal data, we need to be able to ensure *temporal coherence*: Small changes in the data should result in small changes in the visualization; large changes in the data should result in large changes in the visualization. All other mappings of changes in the data to changes in the visualization are arguably bad. Simply put, we want to guarantee that changes perceived by the viewer are due to changes in the data alone. However, while this desiderate is – we argue – clear and evident, there are no treemapping or projection algorithms that comply with it. Even more fundamentally, the very issue of relating data change to visualization change in these two contexts, and deciding what is a 'good' mapping of the former to the latter, is not defined by theory or metrics to gauge it.

1.4 OBJECTIVES AND CONTRIBUTIONS

We have argued that treemaps and projections are valuable tools for making sense of hierarchical and multidimensional data, respectively. However, when it comes to applying these visual encodings to *time-dependent* data, these methods tend to show undesirable traits; and only

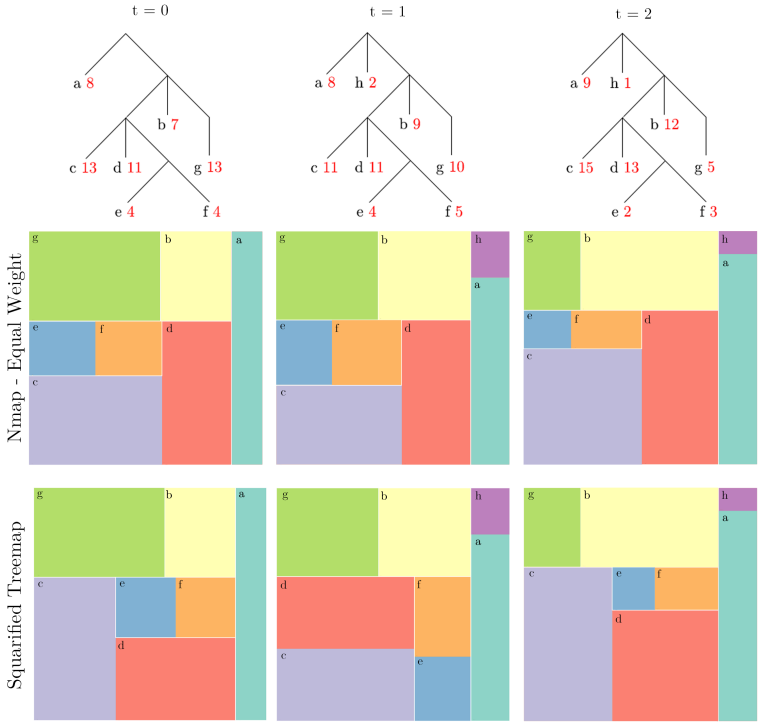


Figure 1.2: Layouts generated by NMap (Duarte et al., 2014) and Squarified Treemap (Bruls et al., 2000) for a sample hierarchical dataset of 3 time steps. We see how NMap is more stable, and similar in aspect ratio, than Squarified Treemap.

limited research effort has been dedicated to understanding, testing, and developing methods that preserve temporal coherence. This research gap leads us to this thesis' high-level research question, stated next as

How to extend projections and treemaps to stably, accurately, and scalably handle temporal multivariate and hierarchical data?

To address this question, there are three components that must be satisfied in either track. We must be able to

A. Develop ways of accurately measuring stability

To evaluate the stability of a dynamic treemap or dynamic projection, we need to have reliable measurement tools that quantify the relationship between data change and visual change.

For treemaps, we developed the *Unavoidable Change* (Vernier et al., 2018b) metric, based on the mathematically proven minimum change that cells would need to undergo to accommodate the data change, and *Baseline Treemaps* (Vernier et al., 2020b), a similar method to approximate the minimum amount of change that any time-dependent treemap must incur when data changes. We also proposed a set of stability metrics for dynamic projections based on the mathematics of visual quality metrics. Before our work, there were no methods designed to measure instability that took into consideration data change – they only looked at visual change, which can be deceiving. As such, we argue that our work provides a contribution to the fundamentals of using treemaps for reliably depicting dynamic hierarchical data.

B. Evaluate methods in the literature considering the tradeoff between stability and visual quality

As already hinted, tens of methods for constructing treemaps and projections exist. However, and as also already hinted, few if none of these methods were gauged from the perspective of stability – one reason thereof being the lack of a *measure* for stability. Having developed such a measure, as mentioned above at point (A), we next use it to produce comprehensive evaluations for dynamic treemaps and projections from the stability perspective. Our contributions in this direction entail work along the following axes:

- *Metrics:* We proposed and implemented a novel set of metrics that reliably measures visual quality and stability.
- *Datasets:* Since there was no previous extensive evaluation or benchmark designed for testing dynamic treemaps or projections, we collected and/or generated a comprehensive collection of datasets that drove our evaluations.
- *Methods:* Collection, implementation, and proposal of several dynamic treemap and projection algorithms. Most of the work on the topic up until our work was conjectural and not quantitatively tested. Simply put, there was no central collection of algorithms that one could use to compare and gauge the performance of dynamic treemaps and dynamic projections. We claim that work solved a large part of this challenge.
- *Analysis:* We combined the previous axes into comprehensive insights into dynamic projections and treemaps. Simply put, we generated quantitative and qualitative evidence showing how existing dynamic projection and treemap algorithms compare to each other, allowing both researchers and practitioners to choose which subset of algorithms are best suited to extend next, respectively directly use in practice.

C. Design state-of-the-art methods that strike a good balance between stability and visual quality

Once the tools necessary to test and compare dynamic treemaps and projections were in place, we were able to leverage the gained insights to produce state-of-the-art algorithms that strike a better balance between stability and visual quality.

Specifically, we have developed Greedy Insertion Treemap (GIT) (Vernier et al., 2018a), a stable and scalable state-aware method for displaying dynamic treemaps. Compared to all treemapping algorithms that we were aware of at the time of GIT's development, we showed that GIT provides a better tradeoff between spatial quality and stability, therefore surpassing its competitors. Moreover, GIT is simple to implement, generic, and computationally scalable.

Regarding projections, we proposed PCD-tSNE and LD-tSNE (Vernier et al., 2021), two neighborhood-based projection methods that use global guides to steer the projected points. This avoids unstable movement that does not encode data dynamics while keeping, at the same time, the highly praised visual quality of the t-Stochastic Neighborhood Embedding (t-SNE) projection method (van der Maaten and Hinton, 2008), arguably the best known high-quality projection method used nowadays for high-dimensional data. As for GIT, PCD-tSNE and LD-tSNE are generic methods, with good computational scalability.

Lastly, it is important to mention that all aspects of our research are open. All the code and data is organized and available online to facilitate further research on dynamic treemaps and projections. This directly supports our answering of the earlier mentioned research questions – which, besides theory, require materials such as data and software for interested researchers and practitioners to replicate, extend, and ultimately use our contributions.

1.5 ORGANIZATION OF THE THESIS

This thesis is structured in two main tracks. The first track covers dynamic treemaps, the challenges related to evaluating such techniques, and the proposal of a new stable algorithm. The second track follows the same organization, with an evaluation of dynamic projection techniques, the introduction of new stable methods, and, finally, an application chapter.

Chapter 2 is an evaluation of treemap methods in the context of dynamic trees extracted from Open Source software repository. In this initial contribution, we propose new ways of effectively quantifying the relationship between data change and visual change. The metrics and concepts presented in this chapter are the base that the whole treemap track is built upon. For this evaluation, instead of using general tree datasets

and having to handle the challenges of data from different sources with a wide range of attributes and traits, for the sake of simplicity, we chose the smaller scope of software evolution datasets.

In Chapter 3, we build upon the foundations of Chapter 2, and present a large scale evaluation of generalized dynamic treemaps. We expand the study in terms of techniques, metrics, and, most importantly, datasets. We use more than 2000 datasets from a wide range of domains, which are next classified according to our novel classification scheme for time-dependent data. This allows us to have a finer understanding of method behavior and how performance relates to the data traits and dynamics.

In Chapter 4, we conclude the treemap track of this thesis introducing Greedy Insertion Treemaps, a novel (stateful) method that jointly optimizes spatial quality and stability. Note that GIT was proposed before the publication of Chapter 3, so, even though in this chapter we only benchmark GIT on software repository datasets, its results for generalized temporal tree data are also available in Chapter 3, which supports its status as a state-of-the-art dynamic treemapping method.

Chapter 5 starts the projection track of the thesis. Following the approach of Chapters 2 and 3, we present an extensive evaluation of dynamic projection algorithms. For such an evaluation, we collect/implement/modify a set of projection methods, we gather time-dependent multidimensional datasets, and we introduce new suitable ways to define and quantify the stability of dynamic projection methods.

Given that the results of Chapter 5 reveal no "best" method for both visual quality and stability, in Chapter 6 we propose two new algorithms that strive to strike a better balance between the aforementioned metrics, making them better suited methods for dynamic projections. We propose PCD-tSNE and LD-tSNE, which use global guides to steer projection points in an attempt to avoid unstable movement while keeping t-SNE's neighborhood preservation ability.

In Chapter 7, we present a real-world application for the dynamic projection methods introduced in Chapters 5 and 6 in the context of hyperkinetic movement disorder analysis. These disorders manifest as abnormal involuntary movements that highly affect the quality of life of the people who suffer from them, and computer supported diagnosis is desired given the complexity of their manifestation.

Finally, Chapter 8 summarizes the work conducted in this thesis, discusses the strengths and weaknesses of our proposed approaches, and suggests directions for future work.

2

TREEMAP EVALUATION FOR SOFTWARE EVOLUTION DATA

As outlined in Chapter 1, one of our two research questions concerns how to extend treemapping algorithms to handle time-dependent hierarchical data. To do this, we need first and foremost to understand how existing treemapping algorithms fare when displaying dynamic data. In more detail, we also need to know how to quantitatively compare such algorithms. In this chapter, we address the above two questions by a first evaluation. For this, we consider a smaller scope in terms of datasets, focusing on dynamic hierarchies coming from evolving software repositories. Using this limited scope of datasets, and beyond evaluating existing treemapping algorithms, we introduce a stability metric for gauging the performance of such algorithms in the presence of time-dependent data, that quantifies the relationship between data change and change in the data's visualization using treemaps. We next extend this approach in Chapter 3 to reliably and fairly test treemapping algorithms on generalized data, i.e., *a large number of datasets from different sources, displaying a variety of traits and dynamics, beyond dynamic hierarchies coming from software evolution.*

Abstract: Dynamic treemaps are one of the methods of choice for displaying large hierarchies that change over time, such as those encoding the structure of evolving software systems. While quality criteria (and algorithms that optimize for them) are known for static trees, far less has been studied for treemapping dynamic trees. We address this gap by proposing a methodology and associated quality metrics to measure the quality of dynamic treemaps for the specific use-case and context of software evolution visualization. We apply our methodology on a benchmark containing a wide range of real-world software repositories and 12 well-known treemap algorithms. Based on our findings, we discuss the observed advantages and limitations of various treemapping algorithms for visualizing software structure evolution, and propose ways for users to choose the most suitable treemap algorithm based on the targeted criteria of interest.

2.1 INTRODUCTION

Hierarchies play a central role in understanding large software systems. Such systems evolve over hundreds of revisions or more, and can have

This chapter is based on the paper "Quantitative Comparison of Dynamic Treemaps for Software Evolution Visualization" (Vernier et al., 2018b)

thousands of elements or more, which are typically organized hierarchically (*e.g.* in folders, files, classes, and methods). Hence, tools for visually understanding evolving hierarchies are a key component in the program comprehension arsenal. Treemaps are a well known method for visualizing hierarchical data. Given an input tree whose leafs have several attributes, treemaps recursively partition a 2D spatial region into cells whose area, color, shading, or labels encode the tree's data attributes. Compared to other methods such as node-link (Harel and Koren, 2002; Frick et al., 1995) or Sunburst (Clark, 2006; Telea et al., 2009) techniques, treemaps use all available screen pixels to show data and thus can handle trees of tens of thousands of nodes.

Dynamic treemaps leverage the above advantages to show dynamic, or evolving, trees. Given a tree sequence, they create an animated sequence of treemap layouts that reflect how the structure and attributes of the trees in the sequence change in time. Evolving treemaps have been created both by using classical static treemap algorithms (Schulz, 2011) or by specialized algorithms (Sondag et al., 2017; van Hees and Hage, 2017; Hahn et al., 2014).

Evolving treemaps have received great interest in software visualization (Diehl, 2007; van Hees and Hage, 2017; Hahn et al., 2014; Fisher and Sud, 2010; Gotz, 2011). As many treemap techniques exist, the question emerged of how to measure their quality. For common rectangular treemaps, which map tree nodes to rectangles, visual quality is typically measured by the aspect ratio of these rectangles. However, the aspect ratio may not capture all desirable qualities of such treemaps. For example, bad aspect-ratio cells of a tiny area could influence the overall visual quality far less than large bad aspect-ratio cells. Atop visual quality, evolving treemaps are assessed by measuring their rate of visual change. However, this metric may not capture all desirable properties: Large visual changes in a treemap are expected (and actually desirable) when the underlying tree changes drastically, but undesired when the tree changes only slightly.

Although treemaps are used for over two decades in software visualization (Shneiderman, 1992; Schulz et al., 2011; Schulz, 2011; von Landesberger et al., 2011), there are few comprehensive evaluations of the quality of dynamic treemap techniques and, to our knowledge, none that focuses on trees capturing software evolution. The aim of this chapter is to fill this gap. For this, we first review the related work in (dynamic) treemaps and their quality measurement, with a focus on software visualization (Sec. 2.2). We next refine desirable treemap properties into 5 quality metrics that capture both spatial quality and dynamic quality (Sec. 2.3). We measure these metrics on 12 well-known treemap algorithms on 28 tree sequences, ranging from a few hundred to tens of thousands of elements, all extracted from software repositories. We next visualize and analyze our results to address questions that practitioners would like to answer to choose a suitable technique (Sec. 2.4). We dis-

cuss our findings and proposed methodology in Sec. 2.5. Our results (datasets, metrics, treemap implementations, evaluation results, and visualizations thereof) are publicly accessible for researchers in the software visualization field interested in evaluating treemap methods for evolving software hierarchies.

2.2 BACKGROUND

Hierarchies are arguably the central element in most software visualizations. They capture the physical (*e.g.* files and folders) or logical software (*e.g.* syntax tree) system structure, together with static or dynamic attributes, *e.g.*, code size, quality metrics (Lanza and Marinescu, 2006), change requests, or testing results (Diehl, 2007). Both static and dynamic hierarchies in program comprehension are typically extracted by mining software repositories (Lanza and Ducasse, 2003; Kagdi et al., 2003). When small (a few hundred nodes), such trees can be visualized using classical node-link layouts such as in class or architecture diagrams (Müller and Klashinsky, 1988; Lanza and Ducasse, 2003; Telea et al., 2002). This works well for architecture-level views on a software system. However, code-level views, which contain nodes from subsystems all the way to classes and methods, generate large trees, having hundreds of thousands of nodes (Telea et al., 2009). These require space-filling methods, such as icicle plots (Holten, 2006; Cornelissen et al., 2007) or, the method of choice, treemaps. The latter are discussed below.

2.2.1 *Treemap algorithms*

Let $T = \{n_i\}$ be a tree with nodes n_i, and let $a_i \in \mathbb{R}^+$ be an attribute defined on the tree leaves. For non-leaf nodes n_i, a_i equals the sum of the attributes of the children of n_i. A rectangular treemap algorithm TM creates a set of rectangle cells $\{c_i\} = TM(T)$, $c_i \subset \mathbb{R}^2$ for the nodes n_i so that the area of c_i equals a_i and children node cells create a partition of their parent cell. Several treemap algorithms exist, as follows (for detailed surveys, see Schulz et al. (2011); Shneiderman and Plaisant (2017); Schulz (2011); von Landesberger et al. (2011)). Slice and dice (SND) treemaps pioneered the concept but were found to create too long-and-thin cells which are hard to grasp (Shneiderman, 1992). Subsequent algorithms tried to improve this aspect, quantified by the aspect ratio (AR) of the treemap cells. Squarified treemaps (SQR) propose a slicing heuristic that achieves, in general, very good (close to one) AR values (Bruls et al., 2000). Nagamochi and Abe refined this idea in an algorithm (APP) that approximates the optimal AR a given treemap can reach (Nagamochi and Abe, 2007). However, SQR is not particularly *stable* – small changes in the input tree can yield large

changes in the treemap layout. Several algorithms have aimed to improve stability. Ordered treemaps (OT) (Shneiderman and Wattenberg, 2001) and Strip treemaps (STR) (Bederson et al., 2002) layout cells c_i to follow a predefined order of the nodes n_i. Different algorithms propose different orderings: Pivot-by-Middle (PBM), Pivot-by-Size (PBZ), and Pivot-By-Split-Size (PBS) (Shneiderman and Wattenberg, 2001); Engdahl's Split algorithm (Engdahl, 2005); and laying out cells along a space-filling curve, *e.g.*, Spiral (SPI) (Tu and Shen, 2007), and Hilbert (HIL) and Moore (MOO) fractal curves (Tak and Cockburn, 2013). Spatially-Ordered Treemaps (SOT) (Wood and Dykes, 2008) extend SQR by ordering sibling nodes so that the most similar ones are processed in turn. NMap (Duarte et al., 2014) uses a related idea; cells are placed according to the similarity of their attributes, using a dimensionality-reduction approach. Two versions exist: NMap Alternate Cuts (NAC) alternate horizontal and vertical cuts to subdivide the space (akin to SND), while NMap Equal Weights (NEW) splits the space to create similar-size cells of similar. However, NMap was only applied to single-level trees. Recently, Sondag *et al.* propose stable treemaps (Sondag et al., 2017), which aim to improve both the AR and stability for dynamic treemaps by using non-sliceable layouts.

Other cell shapes can be used besides rectangles. Voronoi treemaps (Balzer and Deussen, 2005; Balzer et al., 2005) exploit the properties of weighted Voronoi diagrams to create organic-looking displays where cells are convex polygons with, in general, good AR values. Voronoi methods have also been used, with good results, to construct dynamic treemaps for visualizing software structure evolution (van Hees and Hage, 2017; Gotz, 2011). Hybrid treemaps (HTM) (Hahn and Döllner, 2017) combine various basic treemap techniques to generate the final layout. Other variants include jigsaw treemaps (Wattenberg, 2005), orthoconvex treemaps (Berg et al., 2014), and bubble treemaps (Görtler et al., 2018).

2.2.2 *Treemap quality metrics*

In practice, the quality of treemaps is measured using two types of metrics, as follows.

Spatial quality metrics capture how easy one can read the information shown in a static treemap. Such metrics include the aspect ratio (AR) of the treemap cells, which ideally should equal one. For ordered treemaps, the readability metric measures how often one switches visual scanning direction while reading the treemap in order (Bederson et al., 2002); and the continuity metric measures how often cells for neighbor nodes (following the given node order) are not neighbors in the treemap layout (Tu and Shen, 2007).

Stability metrics capture how easy one can follow the changes in a dynamic treemap. Given two treemaps for two (typically consecutive) time-moments t_i and t_j, Shneiderman and Wattenberg (2001) define stability as the distance between the vectors $(x_k(t_i), y_k(t_i), w_k(t_i), h_k(t_i))$ and $(x_k(t_j), y_k(t_j), w_k(t_j), h_k(t_j))$, where x and y are the coordinates of the top-left corner, and w and h, the width, and the height of a cell c_k, averaged over all cells in the treemap. Hahn et al. (2014) use for stability the change of distance between the centroids of $c_k(t_i)$ and $c_k(t_j)$, averaged over all cells. Tak and Cockburn (2013) use the same cell-change metric (top-left corner, width, height) as Shneiderman and Wattenberg (2001), but aggregate via variance rather than average. They also propose a drift metric which measures how much a cell moves away from its average position over a time period. Two recent metrics measure stability at the level of pairs of cells rather than individual cells. Hahn et al. (2017) propose the relative direction change, which measures the angle change of centroids for every pair of cells in a layout. Sondag et al. (2017) measure the relative position change of each cell with respect to eight planar zones defined by four lines given by the edges of that cell, averaged over all treemap cells.

2.2.3 *Software visualization challenges*

Summarizing, considerable effort went into designing static treemap methods and measuring their quality. Less effort went to evaluating dynamic treemaps. We identify limitations in several directions, with a focus on our use-case of visualizing large evolving software hierarchies:

Algorithms: Treemap papers typically compare a few (2..5) algorithms from the much larger set of available ones. In particular, it is not clear how most existing static treemap algorithms perform on the types of dynamic trees extracted from software evolution analyses.

Datasets: Existing methods are typically evaluated on one or a few datasets. While in this chapter, we cannot (and do not aim to) cover the full space of all possible trees, we can do better than current work: For our specific context of software visualization, we aim to know how treemap methods perform on a representative collection of software hierarchies capturing software evolution.

Metrics: As outlined in Sec. 2.2.2, stability is currently measured by looking at how much two treemaps (typically for consecutive time moments) do change with respect to each other. However, when the underlying tree sequence changes a lot, *e.g.* by insertions or deletions of many files or classes at the same moment during a software repository's evolution (an event well-known to take place often in software evolution), the treemap will change a lot, so its evolution

will be labeled as unstable. However, it is actually *desirable* to have a large visual change in this case, as this correctly shows the presence of a large data change. We argue that ways to measure stability as a function of the data change is needed.

Result exploration: Most evaluations consider only aggregated metrics with one value per technique or per technique-and-dataset. Analyzing the actual distribution of metric values over both layout-space and time can give extra insights into the strengths and weaknesses of specific techniques.

Replicability: Treemap evaluations can be hard to replicate as datasets and algorithm implementations are not always openly available or not integrated to make a comparison on different datasets, and along different metrics, easy. Replicability is a growing concern in information visualization but with particular weight in software visualization (Sensalire et al., 2009; Seriai et al., 2014; Merino et al., 2018).

The remainder of this chapter is dedicated to addressing the above points.

2.3 MEASURING THE QUALITY OF DYNAMIC TREEMAPS

To address the current limitations of dynamic treemap evaluations in software visualization, we performed an in-depth study covering the five directions in Sec. 2.2.3, as follows.

2.3.1 *Algorithms*

We consider in our evaluation 12 methods: Approximate (APP), Hilbert (HIL), Moore (MOO), NMap-Alternate-Cuts (NAC), NMap-Equal-Weights (NEW), Pivot-by-Middle (PBM), Pivot-by-Size (PBZ), Pivot-by-Split-Size (PBS), Slice-and-Dice (SND), Spiral (SPI), Squarified (SQR), and Strip (STR) treemaps. For NMap, we use as seed layout the one computed by SQR (for details, see Duarte et al. (2014)). We do not consider non-rectangular treemap methods, as their quality is less easy to compare with rectangular ones, and are also less used in practice. Also, we do not consider the stable treemaps in (Sondag et al., 2017) as this method is considerably slower (over one order of magnitude) than the above-mentioned methods.

2.3.2 *Datasets*

We evaluate all above treemap methods on a collection of 28 datasets (Tab. 1). All of them consist of trees describing the hierarchy of public and well-known GitHub software repositories (folders, files, classes),

Dataset	Revisions	Nodes (total)	Average depth
animate.css	50	3454	2.87
AudioKit	22	11178	6.95
bdb	62	2658	3.83
beets	106	9844	3.75
brackets	88	120292	12.85
caffe	44	12969	4.93
calcuta	50	2882	10.76
cpython	321	584821	6.50
earthdata-search	46	18539	6.82
emcee	64	1746	3.62
exo	97	36436	11.88
fsharp	69	22906	7.89
gimp	72	170418	5.19
hospitalrun-frontend	38	16759	5.71
Hystrix	61	15530	13.29
iina	74	6849	4
jenkins	137	277185	11.94
Leaflet	84	13381	4.86
OptiKey	36	9782	6.72
osquery	37	14111	5.75
PhysicsJS	20	2022	4.6
pybuilder	53	5457	7
scikitlearn	88	48468	5.75
shellcheck	53	746	2.39
soundnode-app	35	3196	6.88
spacemacs	51	10201	4.96
standard	29	203	2
uws	122	4093	2.76
Totals:	2132	1458036	5.77

Table 1: Software evolution tree datasets used in the evaluation.

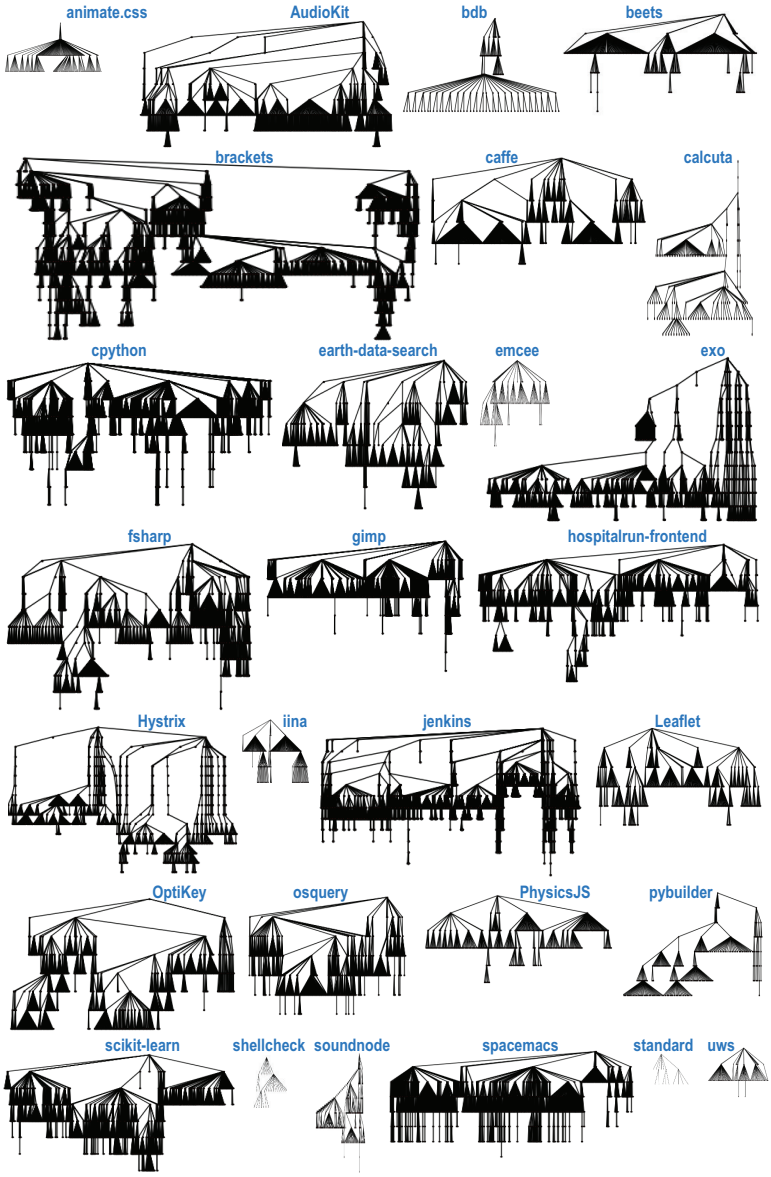


Figure 2.1: Union trees of software evolution tree datasets used in the evaluation. Names correspond to public repositories on GitHub.

one tree per revision, where leaves (classes) are attributed by their number of lines of code. The trees and their attributes have been extracted from the actual repositories by a fully automatic pipeline we built using *libgit2* (Steinhardt, 2018) for repository parsing and Understand (SciTools, 2017) for code analysis. For a more detailed description of the extraction pipeline, we refer to da Silva et al. (2016). The respective software projects have widely different sizes, tree depths and structures, durations, numbers of contributors, language (C, C++, Java, Python), and code type (library, framework, application). This is seen in the figures in Tab. 1 and also in Fig. 2.1 which shows the union trees $\cup_i T(t_i)$ for the considered datasets. Hence, we argue that this collection covers reasonably well the space of tree sequences obtained from software evolution.

2.3.3 *Metrics*

Let w_k and h_k be the weight and height of cell c_k; and (W, H) the width and height of the screen space we draw the treemap in. With these, we consider the following metrics.

2.3.3.1 *Spatial quality metric*

We first consider the classical aspect-ratio metric

$$Q_k^{AR} = \min(w_k, h_k)/\max(w_k, h_k). \tag{2.1}$$

Introduced in Bruls et al. (2000), this metric has been since then used all treemap evaluations to capture spatial quality. As such, we keep it in our evaluation. It is designed to give high scores for rectangles with sides of similar length, and low scores otherwise.

2.3.3.2 *Stability metrics*

Let $c_k(t_i)$ and $c_k(t_j)$ be two cells in two consecutive versions $T(t_i)$ and $T(t_j = t_{i+1})$ for the same node in a dynamic tree. Typical stability metrics (Sec. 2.2.2) only measure the *visual* change δc_k between $c_k(t_i)$ and $c_k(t_j)$. We use for δc_k the average sum of distances between the four corresponding corners of $c_k(t_i)$ and $c_k(t_j)$ (Shneiderman and Wattenberg, 2001), normalized by the treemap diagonal $\sqrt{W^2 + H^2}$, so $\delta \in [0, 1]$. We next define the *data change* between nodes $n_k(t_i)$ and $n_k(t_j)$ as $\delta a_k = |a_k(t_i) - a_k(t_j)|$, where a_k is the relative weight of n_k at time t_i. If either of $n_k(t_i)$ or $n_k(t_j)$ does not exist, *i.e.*, a node was created or deleted in versions t_i or t_j, we set the respective a_k to zero, which is as if the respective node was depicted by a zero-size cell. We normalize $a_k(t_i)$ by the weight sum of all nodes n_k present at time t_i, so

$\delta a_k \in [0,1]$. With this, we define the stability of a cell c_k in a treemap in several ways. First, we define stability as

$$Q_k^{RATIO} = (1 - \delta c_k)/(1 - \delta a_k). \tag{2.2}$$

When visual changes are proportional to data changes, since both are normalized, Q_k^{RATIO} goes to one. Note that an analogy to Eqn. 2.1, *i.e.*, $Q_k^{RATIO} = \min(\delta c_k, \delta a_k)/\max(\delta c_k, \delta a_k)$ does not work: Eqn. 2.1 is symmetric in width and height. For stability (Eqn. 2.2), we want to assess visual change as a function of data change, and not conversely.
A second way to define stability is by

$$Q_k^{MOD} = 1 - |\delta c_k - \delta a_k|. \tag{2.3}$$

For proportional visual *vs* data changes, $Q_k^{MOD} = 1$.

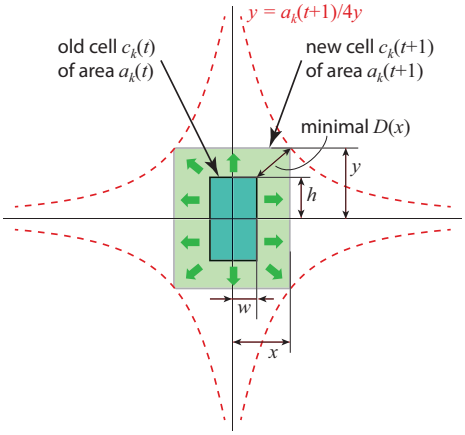


Figure 2.2: Computation of unavoidable change metric Q_k^{UNAV}.

To compare data and visual changes, Q_k^{RATIO} and Q_k^{MOD} must be normalized to the same range, therefore we clip Q_k^{RATIO} to the $[0,1]$ interval. However, this can introduce normalization biases, *e.g.* when the data changes and visual changes have very different ranges. To address this, we next propose to define stability purely in visual space. For this, we consider the actual change δc_k of a cell *vs* the *unavoidable*, *i.e.* minimal, change Δc_k that c_k would need to undergo to accommodate the data change from $a_k(t)$ to $a_k(t+1)$. If $\delta c_k > \Delta c_k$, the algorithm is unstable; if $\delta c_k = \Delta c_k$, it is fully stable. We compute Δc_k as follows (Fig. 2.2). Let $c_k(t)$ be a cell of width w and height h at time step t. Let $c_k(t+1)$ be the version of $c_k(t)$, of area $a_k(t+1)$, at step $t+1$. We first note that δc_k is minimal when $c_k(t)$ and $c_k(t+1)$ have the same center, as visual change is then caused *purely* by data change and not by avoidable 'drift' of the cell corners. Taking a xy coordinate frame centered in this common cell center, the top-right corner of $c_k(t)$ is constrained to

a hyperbola $y = a_k(t+1)/4x$. Hence the minimal change Δc_k is four times the minimal distance D from this corner to the hyperbola, *i.e.*

$$D(x) = \sqrt{(x - w/2)^2 + (a_k(t+1)/4x - h/2)^2}.$$

To find the minimum of D, we solve $\frac{dD^2}{dx} = 0$ for $x \geq 0$. This quartic equation in x has analytic solutions. We obtain x, the width of the optimal cell $c_k(t+1)$, and thereby the minimal Δc_k. Finally, we define the unavoidable-motion stability as

$$Q_k^{UNAV} = 1 - (\delta c_k - \Delta c_k). \tag{2.4}$$

Finally, we define stability for a whole tree T as the *absolute* value of the Pearson correlation coefficient

$$Q^{CORR} = \left| \frac{\sum_k (\delta c_k - \overline{\delta c_k})(\delta a_k - \overline{\delta a_k})}{\sqrt{\sum_k (\delta c_k - \overline{\delta c_k})^2} \sqrt{\sum_k (\delta a_k - \overline{\delta a_k})^2}} \right| \tag{2.5}$$

of the signals $\{\delta c_k\}$ and $\{\delta a_k\}$ for all cells $c_k \in T$, where $\overline{\delta c_k}$ and $\overline{\delta a_k}$ are the signals' averages, so $Q^{CORR} \in [0, 1]$. If visual and data changes δc_k and δa_k are linearly correlated, Q^{CORR} reaches one. Q^{CORR} close to zero indicates uncorrelated changes, *i.e.*, instability.
Compared to existing treemap stability metrics (Shneiderman and Wattenberg, 2001; Hahn et al., 2014; Tak and Cockburn, 2013; Sondag et al., 2017), all our above metrics consider the *relation* of visual change δc_k to data change δa_k. This is a fundamental difference: A treemap method $TM(T) = \{c_i\}$ is a *function* from trees T to cell-sets $\{c_i\}$, so its stability should be defined akin to Cauchy or Lipschitz continuity, which relate function-value ($\{c_i\}$) changes to variable (T) changes rather than measuring function changes only. Indeed: If a function strongly changes, the function *itself* is not necessarily unstable; this can happen when the input variable strongly changes.

2.3.3.3 *Metric weighting*

As mentioned in Sec. 4.1, very small but bad aspect-ratio cells may not strongly influence the overall perceived spatial quality of a treemap, since they are barely visible. The same argument could be made for very small unstable cells *vs* the overall perceived stability. To model these, when computing the average value of the metrics Q^{AR}, Q^{RATIO}, Q^{MOD}, and Q^{UNAV}, we weigh the respective per-cell values Q_k^{AR} (and the other three ones) by the sizes a_k of their cells. We used such weighted metrics in all experiments described next in Secs. 2.4.2-2.4.4. However, the obtained results showed that the aggregated weighted metric values differ only very slightly from their unweighted versions. As such, in the following we will only consider the unweighted metric versions.

2.4 RESULT EXPLORATION

We measure the five metrics (Eqns. 2.2-2.5) on all 28 test datasets (Sec. 2.3.2) processed by all 12 treemap methods (Sec. 2.3.1). We record metrics at the *cell* level (except Q^{CORR}, recorded at tree level). This yields a high-dimensional-and-hierarchical dataset, conceptually a table with seven columns (5 metrics, algorithm ID, dataset ID, time step) and as many rows as the number of measured cells in all datasets, all timesteps. Exploring this data space is a challenge in itself. As noted in Sec. 2.2.3, current treemap evaluations typically present only a few metrics, aggregated to a single (typically average) value per algorithm or per algorithm-and-dataset. To get more insight, we propose several visualizations that present various aspects of the evaluation data to answer specific questions concerning the evaluated algorithms. We proceed in a bottom-up fashion: We first explore the data at the finest (cell) level-of-detail (Sec. 2.4.1). This shows subtle differences between different methods (we show all table rows), but cannot show all evaluated metrics (table columns). Next, we study the quality as a function of time, for one given evolution sequence (Sec. 2.4.2). Thirdly, we compare the aggregated 5 metrics for all dataset and algorithm combinations (Sec. 2.4.3). Finally, we aggregate all results to present a compact comparison of all algorithms (Sec. 2.4.4).

2.4.1 *How does visual change relate to data change (Q1)?*

Before actually evaluating stability, we want to study the distribution of visual changes created by the tested algorithms as function of the respective data changes for all datasets, all timesteps. For this, we show a scatterplot per algorithm (Fig. 2.3), where, for all datasets, x maps $\delta a_i(t_j)$, *i.e.* data change of all cells c_i from time step t_j to t_{j+1}, for all time steps j; and y maps $\delta c_i(t_j)$ (see Sec. 2.3.3.2). A point is thus a cell in a revision of a dataset. To account for overplotting, we compute density maps from these scatterplots using kernel density estimation (Parzen, 1962) and color-code the density using a heat colormap. Ideally, the visual change should be proportional to data change (Sec. 2.3.3.2), so our scatterplots should be close to a diagonal line. We see that this is not the case. All plots show an upwards-pointing 'tail' close to the origin. This tells that most cells with small data changes have disproportionately large visual changes, so instability affects more the small than the large cells. Shallower tails indicate more stable methods, *e.g.* SND. To get a more summarized insight, we also plot a linear-regression line (red), characterized by the slope (α) and the y-intercept (β), and compute the linear correlation coefficient (r) and standard error (s_e) of the points. Larger r coupled with small s_e values indicate methods which correlate visual change with data change better, *e.g.* SND and NAC. We

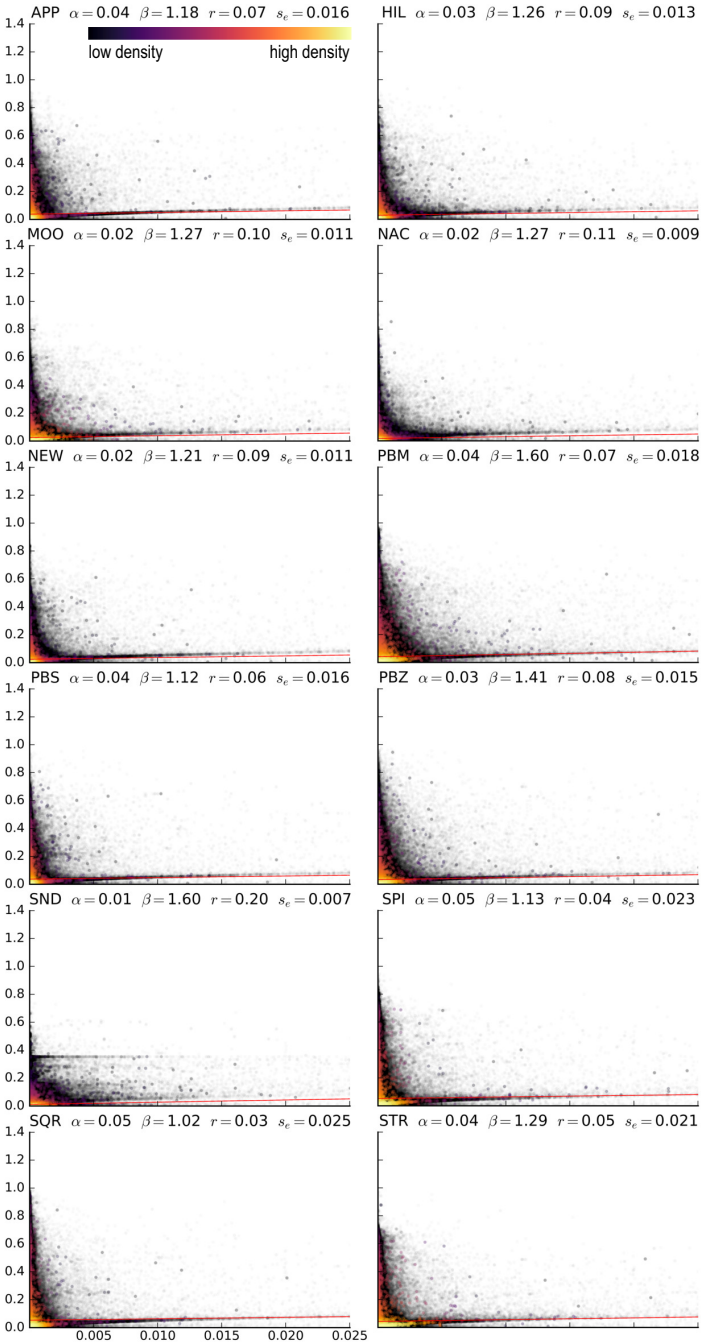


Figure 2.3: Correlation of data and visual change per algorithm, all datasets.

also find the worst-correlating methods, SQR and PBS, and see that SQR is about 7 times worse than SND.

2.4.2 *How is quality evolving in time (Q2)?*

Q1 does not show how quality fluctuates over time for a given tree sequence. Knowing this is important to assess what one can expect when using a given treemap algorithm for a sequence of hundreds of revisions extracted from a repository. To assess this, we show a chart per method, per dataset, and per metric family (that is, spatial quality Q^{AR} and per-timestep averaged values of the four stability metrics Q^{RATIO}, Q^{MOD}, and Q^{UNAV}). In all charts, x maps time and y shows a box plot indicating median (black), 25-75% range (green), and 5-95% range (gray). Since we cannot show this chart for all our 28 datasets (nor can we aggregate them in a single chart), we select one representative dataset to depict: *cpython*. The dataset was extracted from the official Github repository hosting the source code of the Python programming language (van Rossum, 2017). This is our largest dataset with 321 revisions and an average of over two thousands tree nodes per revision. Results for other datasets can be found online (The Authors, 2017).

Figure 2.4a shows the evolution of the Q^{AR} metric (Eqn. 2.1) for all tested methods for *cpython*. We see that APP and PBS deliver overall quite high and constant-over-time aspect ratios (0.7), so they are the best methods for spatial quality, with APP being better as it has a narrower Q^{AR} spread around a slightly higher median value. SQR scores higher median values, but has a larger spread – for every revision, it can score as bad as 0.05 aspect-ratio, while APP does not drop below 0.4 (compare the bottoms of the gray bands in Fig. 2.4 for APP and SQR). SND shows the worst spatial quality, with a tight spread around a median Q^{AR} below 0.1. The chart also tells us that most methods deliver *consistent* spatial quality regardless of the data changes in the 321 revisions (which we found to be large by manually examining the sequence). The quality decrease shown by SND and (less) by HIL and STR is somehow surprising, as none of the studied methods uses a 'history' of the tree-sequence in its layout heuristics.

Figure 2.5b shows the evolution of four stability metrics Q^{RATIO}, Q^{MOD}, and Q^{UNAV}, averaged per time-step. Compared to spatial quality, we see now much more variation between methods and also much more variation (of the stability) over time. We see that SND is by far the most stable method, whereas SQR, SPI, and PBM score worst. Long 'icicle' like boxplots indicate revisions where much more visual change was present than 'warranted' by the data change. Interestingly, these appear at the same moments for different algorithms (Fig. 2.5b, red markers shows one example). For such moments we see large variations across methods: For SPI, this is the most unstable part of the sequence, both

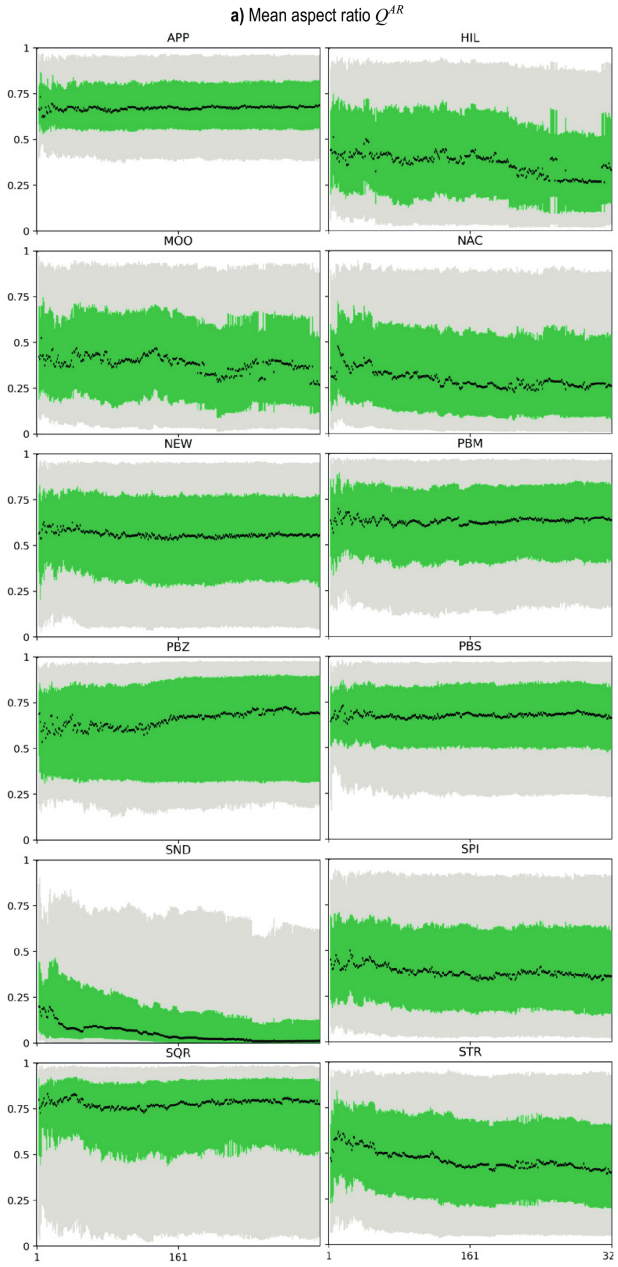


Figure 2.4: Evolution in time of spatial quality (a) for the *cpython* dataset.

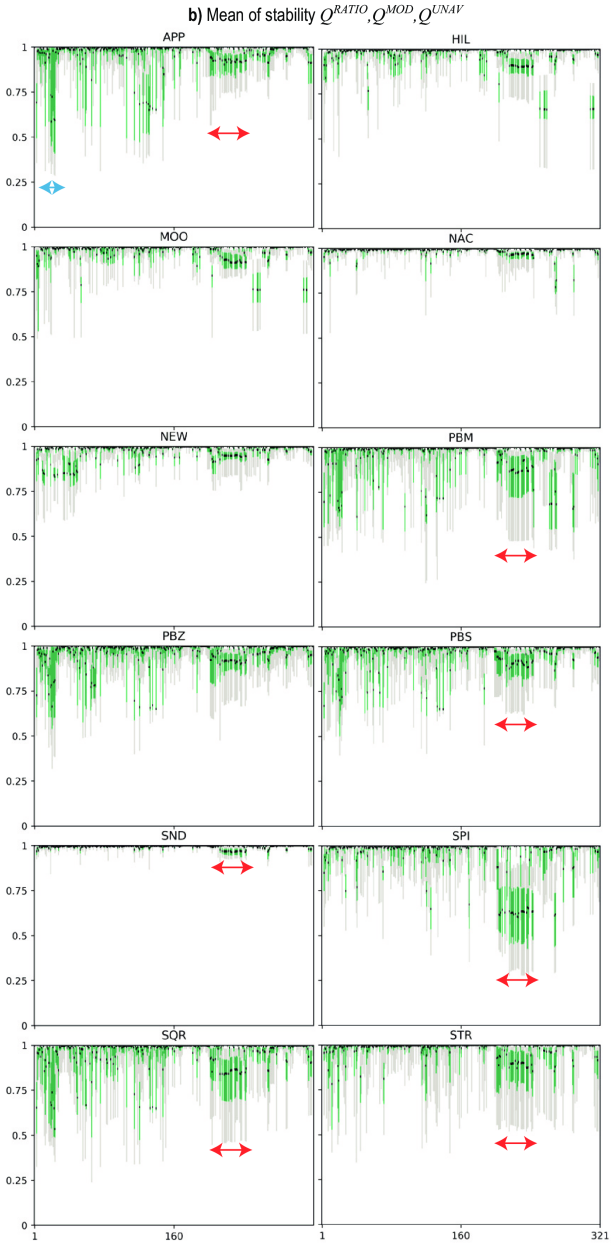


Figure 2.5: Evolution in time of the averaged four stability metrics (b) for the *cpython* dataset.

in median and 5-95% range sense, whereas APP finds earlier sequences (marked in blue in Fig. 2.5) which are harder to lay out stably.

2.4.3 *How do methods perform on different datasets (Q3)?*

So far, we presented charts aggregate over all datasets (Sec. 2.4.1) or focus on a single dataset, but aggregate all stability metrics (Sec. 2.4.2). We would like to see how the proposed stability metrics compare to each other, as we are still in the process of understanding their measurement characteristics. Also, we would like to see how these metrics vary over several datasets. For these goals, we use a set of table views, one per quality metric. In each table, columns are datasets and rows are algorithms, respectively. Each cell thus encodes the average value of one quality metric for one dataset tested by one algorithm. Cells are colored with a luminance-based colormap, with data values separately normalized per metric table, so that darkest cells indicate worst cases in all tables (but with potentially different metric absolute values), and brightest cells indicate best cases in all tables, respectively.

Figure 2.6 tells several interesting things. Scanning the first table rowwise, we see that there are no large aspect-ratio quality differences between the tested datasets. This tells that most methods (with the notable exception of SND) achieve quite good aspect ratios for a wide dataset variation. Over all datasets, APP is the best method, surpassed by SQR only for a few datasets. Conversely, we see that SND is the most stable method with respect to all four considered stability metrics. Stability-wise, we see that some datasets (*hospitalrun-frontend*, *Leaflet*, and *PhysicsJS*) consistently score worse than all others for basically all algorithms. These are also the datasets yielding the worst stabilities, when PBM, SQR, STR, and SPI methods are used. This indicates that these methods are quite sensitive in stability on the type of input dataset so, for obtaining higher stabilities, other methods should be used. At a higher level, we see that the Q^{RATIO}, Q^{MOD}, and Q^{UNAV} stability metrics yield very similar plots. This is an interesting findings, since the metrics have quite different formulations (Sec. 2.3), and indicates that the results can be trusted – the chance of three metrics having such different expressions yielding so similar values being very small. In contrast, the Q^{CORR} metric has much lower values, which is explained by the fact it is much more conservative – a good algorithm would need to yield very well correlated δa_k and δc_k values, and we have seen in Sec. 2.4.1 that this is by far not the case. We conclude that visual *vs* data change correlation is a too strong quality desiderate for dynamic treemaps handing large real-world datasets, and advise next to use in practice any of the Q^{RATIO}, Q^{MOD}, and Q^{UNAV} metrics to gauge stability, or, as we have done in Sec. 2.4.3, their average value.

Figure 2.6: The five quality metrics for all tested methods, all datasets.

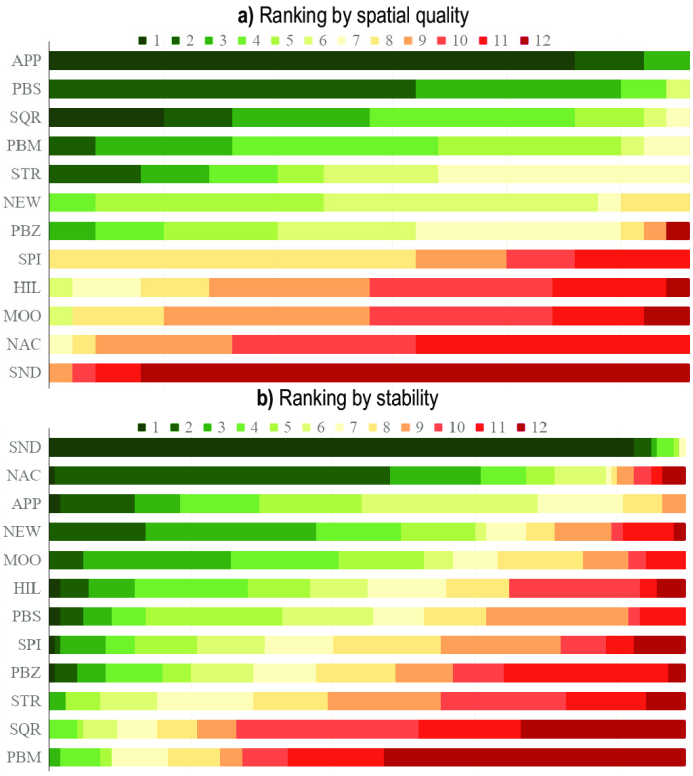


Figure 2.7: Ranking of the 12 methods showing the percentage of times they scored a certain rank with respect to spatial quality (a) and averaged stability (b).

2.4.4 *How to summarize the comparison (Q4)?*

The visualizations so far (Q1..Q3) have given us several insights: We have seen that APP, PBS, and SQR are the best methods with respect to spatial quality, while SND performs poorly for that, but it is the best for stability; different methods have quite different spreads of quality over a given tree sequence, some delivering more consistent results than others, but for most algorithms do not degrade over time; and several of the proposed stability metrics are strongly correlated. It is now useful to *summarize* our findings to present a compact ranking of the tested methods. For this, we use two stacked bar charts. Each bar maps one method and is divided into segments. A segment's length tells the percent of the total number of versions (of all datasets) for which that method had a specific rank regarding spatial quality (Fig. 2.7a) and averaged stability metrics (Fig. 2.7b). We color segments by an ordinal colormap to show these ranks (1 being the best and 12 being the worst). Bars (methods) are sorted in each chart to put the one with highest average rank, weighted by the percents of the total number of versions for all obtained ranks, at the top (Fig. 2.7). From Fig. 2.7, we first see that spatial quality and stability are strongly inversely correlated – methods that score well on one tend to score poorly on the other. We also see that the top methods in both charts are very good for *most* of the tested datasets, *i.e.*, it is easy to find a method that optimizes either spatial quality or stability, but not both. Interestingly, APP (a less known method) is better in spatial quality, and significantly better in stability, than SQR (arguably the method of choice for creating good aspect-ratio treemaps), so it should be preferred to SQR. Similarly, for stability, APP and NEW (two less known methods) are in the top-four most stable methods, and while worse than SND (very well known method), they have higher spatial quality, so they should be preferred to SND.

A disadvantage of the rank charts in Fig. 2.7 is that they do not easily allow linking spatial quality and stability. To alleviate this, we propose a final visualization which uses a start plot metaphor (Fig. 2.8). The scatterplot points (circles, categorically colored) are methods attributed by their average spatial quality and stability over all datasets, all revisions. Each method is linked with the 28 tested datasets by same-color lines; a line's endpoint has the average spatial quality and stability over all its revisions for the corresponding method. The plot conveys several insights: First, methods follow roughly a concave curve (Fig. 2.8, thick dashed curve), telling the trade-off between spatial quality and stability. Variation in average spatial quality is much larger (roughly 45%) than in average stability (roughly 8%). The fan-out of lines from a method shows how predictable that method is, and here we see large variation over methods, with *e.g.* APP being quite consistent in spatial quality, while MOO, STR, and SND show large dataset-dependent variations in both spatial quality and stability (see Fig. 2.8, thin dashed curves). The

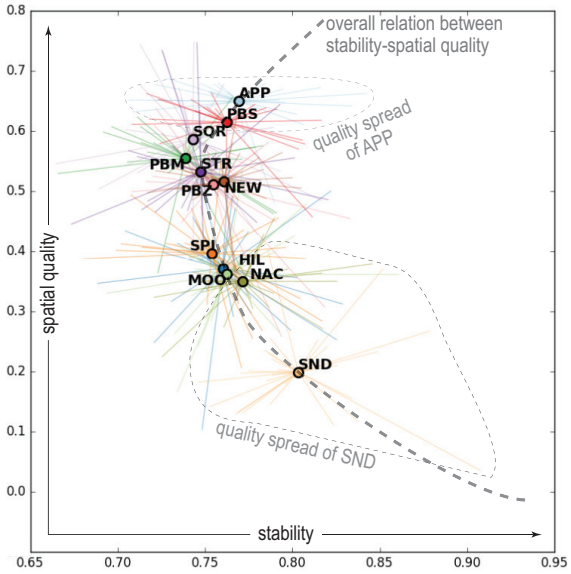


Figure 2.8: Summarized comparison of all methods (colored dots) on all datasets (colored lines) *vs* spatial quality and stability.

latter is especially interesting: Even though SND has the highest average stability, it can also score worse than many other methods on certain datasets.

To conclude, it is hard to designate an 'optimal' method, as this strongly depends on which of stability and spatial quality users see as most important for their concrete use-cases, and by how much. Still, based on all our insights, we believe that APP offers a very good compromise – very high spatial quality and overall stability similar to most methods, surpassed only (and not in all cases) by SND.

2.5 DISCUSSION

Let us discuss our results in the light of the dimensions of evaluating treemap algorithms for software evolution visualization (Sec. 2.2.3):

Algorithms: We consider 12 well-known treemap methods, in contrast to typically 2..3 techniques in current treemap evaluations in software visualization papers. We argue that this gives valuable insights on the suitability of such well-known methods for handling evolving software trees, so it makes the choice of a given method easier for the software visualization practitioner. For instance, our evaluation can tell the interested user which are the advantages (or limitations) of a *given* algorithm *vs* another given algorithm, from the perspectives of spatial

quality or stability.

Datasets: Our treemap benchmark cannot cover all variations of trees extracted from software evolution use-cases. However, it measures in total roughly $1.9 \cdot 10^9$ treemap cells for 28 tree sequences up to 321 time-steps (revisions). The size and variability of tree sequences covered by our study is larger than all existing similar evaluations of dynamic treemaps in software visualization. However, we admit that we cannot *extrapolate* from this evaluation to draw statistically strong conclusions concerning the quality of a given treemap algorithm for the *entire* space of evolving software hierarchies. Doing so would require (a) a characterization of this space in terms of objective metrics (*e.g.*, tree size, depth, type of structure, type of changes); (b) a targeted search of software repositories to extract trees which 'sample' well all these dimensions; (c) an evaluation of our metrics on this benchmark; and (d) most importantly, finding possible correlations between the measured performance of algorithms and the characteristics of the tree sequences they work on. We acknowledge these limitations, and outline them as important directions for future work.

Metrics: We measure treemap stability by essentially considering the first derivative of the treemap algorithm function mapping from tree-node weights to rectangular cell-sets. We detail four variants for measuring stability this way, and observe that three of them, while quite different in terms of actual definitions, yield very similar results. We believe this is an important finding, as it motivates the idea of defining stability by relating visual change to data change. The fourth stability metric (Pearson correlation) showed however to be of limited practical use, as typical dynamic treemaps exhibit a too low correlation of the data and visual changes as compared to other phenomena where this metric is used. This can also indicate that dynamic treemaps may exhibit a more *complex* form of data *vs* visual change correlation than *linear* one. Concluding, we argue that measuring stability by involving both visual and data change is desirable, but we acknowledge that more work is needed to further refine the definition of the proposed stability metric, so that it avoids potential normalization biases, and it also captures in a more demonstrable way what actual users perceive as 'unstable'.

Result exploration: We present five visualizations of treemap quality metrics, covering all involved dimensions: cells, revisions, datasets, metrics, and algorithms. As the dimensionality of this data space is large, we obviously cannot cover *all* possible viewpoints. Yet, our visualizations help finding novel insights on the behavior of dynamic treemaps for evolving software hierarchies, and also confirm earlier observations, *e.g.* the known stability of SND. Our visualizations

can be used to both analyze fine-grained details (at cell level) and present aggregated conclusions (at algorithm level). They can help the practitioner in understanding what is gained, and/or lost, by choosing a certain treemap algorithm instead of another one.

Replicability: All our results (datasets, treemap and visualization code, measurements) are available online at (The Authors, 2017). To our knowledge, this is the first benchmark for (dynamic) treemaps for applications in software evolution understanding. It can serve both for practitioners interested in choosing an algorithm based on specific quality criteria, but also for researchers aiming to benchmark their new algorithms, with limited effort, against existing ones.

Limitations: There are several points which can be better covered better. First and foremost, as mentioned, we need a more principled sampling of the space of trees extracted from software evolution to gain more confidence in the obtained quality results (or how these would differ as a function of the tree sequences' characteristics). We argue that our current work, *i.e.* the *automated* set-up of the extraction pipeline of dynamic trees from software repositories, computation of the proposed quality metrics, and visualizations that aggregate these, forms the necessary basis for such extensions, which we consider as future work. Separately, more treemap algorithms could be considered, *e.g.*, Voronoi, hybrid, or bubble ones. This will require an adaptation of the spatial quality and stability metrics so they can be used for non-rectangular cells.

Threads to validity: Similar to software quality, we measure treemap quality by a number of 'proxy' metrics. While we argue for these metrics at technical level (see the stability metric *vs* function continuity discussion) or, separately, reuse well-known metrics (see the aspect-ratio metric), we do not have hard evidence that such metrics truly capture quality as seen by the eyes of the beholder (end user). The advantage of using such 'intrinsic' quality metrics is that they can be computed automatically, on a large benchmark, and are independent on actual tasks, users, or use-cases. This allows for direct and objective comparisons, parallel to what is done on the context of *e.g.* Graph Drawing (Hachul and Jünger, 2006; Battista et al., 1997), where metrics such as number of crossings, angle of crossings, and distribution of edge lengths are used to rank the quality of graph drawing algorithms. The disadvantage is that we cannot directly infer, from such metrics, how fit to purpose a given treemap technique will be given a specific user, use-case, type of dataset, and task. We argue for our approach as follows: *if* for a given user, use-case, and task, one agrees that a good treemap algorithm should have the properties captured by our quality metrics, *then*

one can use our evaluation and related artifacts (benchmark, metrics, visualizations) to find the best suitable algorithms.

2.6 CONCLUSIONS

We have presented an evaluation of treemap algorithms for the visualizaton of dynamic tree sequences extracted from the evolution of software repositories. For this, we proposed a benchmark formed by 28 datasets extracted from well-known such software repositories, five metrics that aim to capture spatial quality and stability, and 12 known treemap methods. We also propose six visualizations aimed at interpreting the measurement results from several angles, covered by four types of questions. All results (datasets, treemap implementations, measurement code, and visualizations) are publicly available and can constitute the basis of a benchmark for treemap evaluation for visualizing evolving software hierarchies.

Several directions exist for extending this work. First and foremost, a finer-grained analysis of the space of evolving software trees can be made to elicit correlations between characteristics of the datasets and measured quality of the tested treemap methods. Secondly, and at a higher level, it would be useful to extend this type of benchmarking to other application domains that generate dynamic trees and use treemap methods to visualize their evolution in time. This second direction of work will be further explored in the next chapter.

3 GENERALIZED TREEMAP EVALUATION

In the previous chapter, we have presented an evaluation of 12 algorithms designed for construction of static treemaps for the visualization of dynamic hierarchies. The evaluation has outlined that there is no ideal treemapping algorithm that optimally balances stability against visual quality. While these results indicate that better treemapping algorithms could be designed for the visualization of dynamic hierarchies obtained from evolving software systems, an important question is whether we can extrapolate these findings to dynamic hierarchies that emerge from datasets coming from different domains. Additionally, there are more treemapping algorithms, and more ways to gauge their quality, than those which have been considered in the previous chapter. In this chapter, we extend the evaluation from the previous chapter to cover the above directions. First and foremost, we consider dynamic hierarchies produced by a wide range of application domains, and, to address this aspect, we propose a strategy to comprehensively sample the space of time-dependent hierarchical datasets. Secondly, we consider additional quality metrics and ways to compare the performance of the studied algorithms. Finally, we consider a few recent treemapping algorithms that were not used in the evaluation in the previous chapter. Taking all these aspects together, the work presented in this chapter represents the most comprehensive quantitative evaluation of dynamic treemaps presented in the literature.

Abstract: Rectangular treemaps are often the method of choice to visualize large hierarchical datasets. Nowadays such datasets are available over time, hence there is a need for (a) treemaps that can handle time-dependent data, and (b) corresponding quality criteria that cover both a treemap's visual quality and its stability over time. In recent years a wide variety of (stable) treemapping algorithms has been proposed, with various advantages and limitations. We aim to provide insights to researchers and practitioners to allow them to make an informed choice when selecting a treemapping algorithm for specific applications and data. To this end, we perform an extensive quantitative evaluation of rectangular treemaps for time-dependent data. As part of this evaluation we propose a novel classification scheme for time-dependent datasets. Specifically, we observe that the performance of treemapping algorithms depends on the characteristics of the datasets used. We identify four potential representative features that character-

This chapter is based on the paper "Quantitative Comparison of Time-Dependent Treemaps" (Vernier et al., 2020b)

ize time-dependent hierarchical datasets and classify all datasets used in our experiments accordingly. We experimentally test the validity of this classification on more than 2000 datasets, and analyze the relative performance of 14 state-of-the-art rectangular treemapping algorithms across varying features. Finally, we visually summarize our results with respect to both visual quality and stability to aid users in making an informed choice among treemapping algorithms. All datasets, metrics, and algorithms are openly available to facilitate reuse and further comparative studies.

3.1 INTRODUCTION

Treemaps are one of the best-known methods 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. Compared to other methods such as node-link techniques, treemaps effectively use all available screen pixels to show data, and thus can display trees of tens of thousands of nodes on a single screen. Most treemaps use rectangles, although there are alternative models such as Voronoi treemaps (Balzer et al., 2005), orthoconvex and L-shaped treemaps (Berg et al., 2014), and Jigsaw treemaps (Wattenberg, 2005). In this paper, we focus exclusively on rectangular treemaps.

The input for a rectangular treemap is a rectangle R and a set of nonnegative values $a_1, \ldots, a_n$ together with a hierarchy on these values (represented by a tree). 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). Such a partition of a rectangle into a set of disjoint rectangles is also called a *rectangular layout*, or *layout* for short. Typically the input values are *normalized*, that is, the sum $A = \sum_i a_i$ corresponds to the area of R.

Nowadays, large hierarchical datasets are also available over time. Hence, there is a need for *time-dependent* treemaps which display changing trees and data values. Ideally, such time-dependent treemaps enable the user to easily follow structural changes in the tree and in the data. In a time-dependent setting, the input values become functions $a_i \colon [0, X] \to \mathbb{R}_{\geq 0}$ 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-dependent, and that the values a_i are properly 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 t; and we speak of insertions or deletions if $a_i(t)$ starts or stops to be nonzero, respectively.

The *visual quality* of rectangular treemaps is usually measured via the aspect ratio of its rectangles. This indicator 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 (2007) describe an algorithm (APP) which computes, for a given set of values and a hierarchy, a treemap which provably approximates the optimal aspect ratio. Berg et al. (2014) prove that minimizing the aspect ratio for rectangular treemaps is strongly NP-complete. Kong et al. (2010) propose perceptional guidelines to improve treemap design and Zhou et al. (2017) perform user studies to test the effectiveness of different rectangular treemapping algorithms. Recently Lu et al. (2017) argue that the optimal aspect ratio for treemaps should, in fact, be the golden ratio. In Section 3.2, we describe the state-of-the-art of rectangular treemaps in detail along with the various characteristics of rectangular treemaps.

For time-dependent treemaps, a second quality criterion is *stability*. Ideally, small changes in the data should result only in small changes in the treemap. 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. In recent years a few non-rectangular treemaps were specifically developed for time-dependent data. Hahn et al. (2014) and van Hees and Hage (2017) describe stable versions of Voronoi treemaps. Chen et al. (2017) propose a small-multiple metaphor to visualize time-dependent hierarchies. Their algorithm computes a global layout for all time steps simultaneously, but does not handle insertions or deletions. Scheibel et al. (2018) give an algorithm that maps changes in the data onto an initial layout. However, "treemaps" of subsequent time steps are not proper rectangular layouts as white space is introduced when resolving overlaps between rectangles.

A different approach to visualizing time-varying hierarchical data is taken by Lukasczyk et al. (2017), Köpp and Weinkauf (2019), and Li et al. (2019), who show how to compute static overviews of the entire evolution of the tree. Alternatively, Guerra-Gómez et al. (2013) and Card et al. (2006) use interactivity to explore time-varying data. For a broader perspective on tree visualizations, Graham and Kennedy (2010) present a survey of visualizations that compare multiple trees, while Schulz et al. (2011) and Scheibel et al. (2020) present, respectively, a survey and a taxonomy for the visualization of a single tree.

Contribution. Despite their enduring popularity, a comprehensive evaluation of treemaps is currently lacking, even more so for the time-dependent case. Individual papers tend to report on only a few algorithms and evaluate only a few datasets, often without a principled discussion of quality metrics. To provide insights to both researchers and practitioners and to allow them to make an informed choice when selecting a treemap for their specific application and data, we perform

an extensive quantitative evaluation of rectangular treemaps for time-dependent data. Our three main contributions are:
(1) We introduce a new method to measure the stability of time-dependent treemaps which explicitly considers the input data (Section 3.3.2). An algorithm is stable if small changes in the input data result in small changes in the layout, that is, data change and layout change correlate positively. Previously proposed stability metrics measure only the layout change and conclude that small layout changes are a sign of a stable algorithm. However, to properly measure stability, we also need to capture the data change and then correlate data and layout change. 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-dependent treemap must incur (given the input data) when moving from treemap T to the next treemap T'.
(2) We propose a novel classification scheme for time-dependent datasets. Specifically, based on our discussion of the state-of-the-art of treemaps in Section 3.2, we observe that the performance of treemaps depends on the characteristics of the datasets used. We identify four potential representative features that characterize time-dependent hierarchical datasets and classify all datasets used in our experiments accordingly. We experimentally test the validity of this classification on 2405 datasets, and analyze the relative performance of 14 state-of-the-art rectangular treemapping algorithms across varying features. Generally we conclude that our proposed features do indeed have predictive value, both with respect to visual quality and stability. We also observe that algorithms that are designed to be stable tend to in fact be more stable across features.
(3) We perform a quantitative evaluation of 14 rectangular treemapping algorithms on more than 2000 datasets. We visually summarize our results with respect to both visual quality and stability to aid users in making an informed choice among treemaps. All datasets, metrics, and algorithms are openly available (The Authors, 2020a). Section 3.5 reports on our experimental results, we conclude in Section 3.6.

3.2 RECTANGULAR TREEMAPS

We next discuss the most well-known rectangular treemapping algorithms. 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 evaluation. Recall that the input for a rectangular treemap is a rectangle R and a set of non-negative values $a_1, \ldots, a_n$ together with a hierarchy on these values (represented by a tree). The children of a node in this hierarchy are given in a particular order

in the input. We distinguish two classes of treemaps, which either do or do not use this order. For time-dependent data we also distinguish between state-aware and stateless treemaps. Contrary to stateless treemaps, state-aware treemaps do not compute the treemap separately at a time step, but (can) use the layout of the previous time step to compute a new layout. Most treemaps are stateless; we discuss the state-aware algorithms separately.

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**) (Bruls et al., 2000) and Approximation treemaps (**APP**) (Nagamochi and Abe, 2007). 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 worsen visual quality. We include nine ordered treemaps in our evaluation. The first ordered treemaps (Shneiderman and Wattenberg, 2001) include the Pivot-by-Middle (**PBM**), Pivot-by-Size (**PBZ**), and Pivot-By-Split (**PBS**) algorithms. Similar algorithms are the Strip algorithm (**STR**) (Bederson et al., 2002) and the Split algorithm (**SPL**) (Engdahl, 2005). Other algorithms, like the Spiral algorithm (**SPI**) (Tu and Shen, 2007), and the Hilbert (**HIL**) and Moore (**MOO**) algorithms (Tak and Cockburn, 2013), lay out rectangles following a space-filling curve. Finally, the very first treemap algorithm (Slice-and-Dice (**SND**)) by Shneiderman (1992) 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. The treemap for the first time step is typically an existing unordered treemap. The first state-aware treemap was introduced by Sondag et al. (2017). Their Local Moves algorithm (LM) is initialized with APP, and allows only a small number of local modifications to the (combinatorial) layout between time steps. They also show how to update areas between time steps without significantly changing the layout (layouts remain "order-equivalent"). In our evaluation we include the Local Moves algorithm with 4 local moves between time steps (**LM4**), and without local moves (**LM0**). A similar algorithm is the Git algorithm (**GIT**) Vernier et al. (2018a), which is initialized with SQR, and does not allow any changes to the (combinatorial) layout between time steps. Git is described in full detail in Chapter 4, as it is a particular contribution to this thesis. Both state-aware treemaps also support insertions and deletions, updating the layouts locally where necessary (for insertions, the position in the layout can be chosen to maximize visual quality). By design, the stability of state-aware treemaps is relatively unaffected by frequent *weight changes* over time. Also, the visual quality of the initial treemaps should be relatively high. However, since the layouts cannot change much over time, the visual quality of state-aware treemaps will decrease over time if weights change significantly. Frequent *insertions and deletions* may also cause treemaps with poor visual quality, as treemaps are not recomputed as a whole. However, many insertions can help to correct rectangles with bad aspect ratio caused by weight changes over time. This is especially helpful for state-aware algorithms that do not allow any changes to the layout, like LM0 and GIT. 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.

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.

3.3 METRICS

Wattenberg (2005) 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. We also introduce a new method to measure the stability of time-dependent treemaps which captures inherent data changes. We compute metrics for each leaf rectangle separately and then aggregate these values for each algorithm and dataset (see Section 3.5 and (The Authors, 2020a) for details). Note that we do not compute metrics for non-leaf nodes.

3.3.1 *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, visual treemap quality 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 (Lu et al., 2017) 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

$$\rho(R_i) = \min(w(R_i), h(R_i))/\max(w(R_i), h(R_i)). \tag{3.1}$$

Observe that this definition is the inverse of the usual definition for aspect ratio. Its values range from 0 to 1, where values of ρ 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.

3.3.2 *Stability*

Evaluating the stability of a treemap is more involved than evaluating visual quality. Consider treemaps at two consecutive time steps $T(t)$ and $T(t+1)$. Since stability does not *explicitly* depend on the value of t, we denote the former and the new treemap by T and T' respectively, to simplify notation. 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 treemap's layout $d(T, T')$, usually computed by evaluating the change $\delta(R_i, R'_i)$ for each rectangle separately and aggregating it over all rectangles. Shneiderman and Wattenberg (2001) define δ 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.* (Hahn et al., 2014; Hahn, 2015) simplify this metric by defining δ as the distance moved by the centroid of a rectangle, again defining d as the average. Tak and Cockburn (2013) use the same δ as (Shneiderman and Wattenberg, 2001), but define d as the variance over all values computed by δ. They also propose a drift metric, which measures how much a rectangle moves away from its average position over a long period. Recently, Scheibel et al. (2018) introduced two new layout change metrics: The *average aspect ratio change* defines δ 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 δ 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. Chen et al. (2017) propose a metric to quantify the ability of users to track time-dependent data in treemaps, which is closely related to the drift metric (Tak and Cockburn, 2013). A different approach measures layout change using pairs of rectangles. Hahn et al. (2017) introduce the *relative direction change*, which, for every pair of rectangles R_i and R_j, measures how much the angle from the center of R_i to the center of R_j changes. Finally, Sondag et al. (2017) proposed the *relative position change*, which, for every rectangle pair (R_i, R_j), measures how much the relative position of R_i with respect to R_j changes. The distance d is then defined as the average over all rectangle pairs.

Summarizing the above, 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 (Tuytelaars and Mikolajczyk, 2007; Szeliski, 2010). In the vision community, it was established already many years ago (Shi and Tomasi, 1994; Biederman, 1987) 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 (2001). Specifically, let $w(R)$ and $h(R)$ be the width and height of an input rectangle R, respectively. Let p_i, q_i, r_i, and s_i (p'_i, q'_i, r'_i, and s'_i) be the positions of the

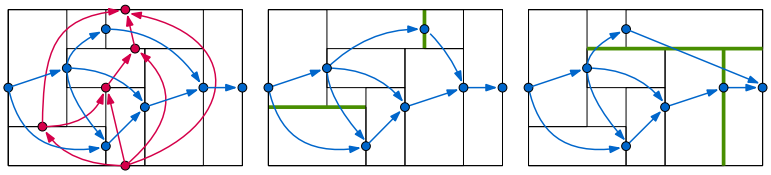


Figure 3.1: Left: Partial orders of the maximal segments. Middle: a layout order-equivalent to the left figure, changed maximal segments highlighted in green. Right: a layout not order equivalent to the other two figures. Red/blue arrows: relations between maximal segments.

corners of a rectangle R_i (R'_i). We define the normalized corner-travel (CT) distance for a rectangle as

$$\delta_{\mathrm{CT}}(R_i, R'_i) = \frac{\|p_i - p'_i\|_1 + \|q_i - q'_i\|_1 + \|r_i - r'_i\|_1 + \|s_i - s'_i\|_1}{4\sqrt{w(R)^2 + h(R)^2}}. \quad (3.2)$$

where $\|x\|_1$ denotes the ℓ_1 norm. Simply put, δ_{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 (Sondag et al., 2017). We established experimentally that the corner-travel and the relative position change metric correlate clearly on more than 2000 data sets. Hence in Section 3.5 we report only on experiments using the corner-travel distance. All other data can be found here (The Authors, 2020a).

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 (Vernier et al., 2018b,a). 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*. For this, we define a *baseline* treemap T^* with respect to T and T'. The layout of T^* (that is, the combinatorial structure of the rectangular subdivision which constitutes T^*) is based on 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-dependent 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, which was proven to converge in Eppstein et al. (2012). For a rectangular layout (treemap) T, a *maximal segment* is a maximal contiguous horizontal or vertical line segment contained in the union of the borders of all rectangles in T (for example, the green segments in Figure 3.1). Put simply, a horizontal maximal segment (which is not part of the input rectangle R) always has endpoints on the interior of two vertical segments and vice versa. For two horizontal maximal segments s_1 and s_2, we say that $s_1 < s_2$ if there is a rectangle in T whose bottom side coincides with s_1 and whose top side coincides with s_2. This defines a partial order on horizontal maximal segments. We define a partial order on vertical maximal segments analogously (Figure 3.1). We say that T is *order-equivalent* to T^* if the corresponding partial orders on maximal segments are isomorphic. For every possible set of areas, there exists an order-equivalent treemap to T that correctly represents those areas Eppstein et al. (2012). In particular, we can initially define T^* as the treemap order-equivalent to T (computed with any of the tested algorithms) with the areas $\{a'_1, \ldots, a'_n\}$ of T'.

If rectangles are inserted or deleted, the baseline treemap 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 3.2), 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 as evenly as possible. To do so, we replace every maximal segment e in T by a rectangle, and assign each such rectangle a portion of the inserted area corresponding to the length of e (Figure 3.2). Hence all walls become equally thick and the original rectangles of T need to move as little as possible to yield T^*.

The baseline treemap T^* as proposed here is not a perfect baseline, as it does not always minimize the movement of every rectangle. However, the layout change between T and T^* is still a very good estimate for the minimum necessary layout change between T and T', and thus a good

measure for data change (see Figure 3.3: nearly all points lie on or below the diagonal).

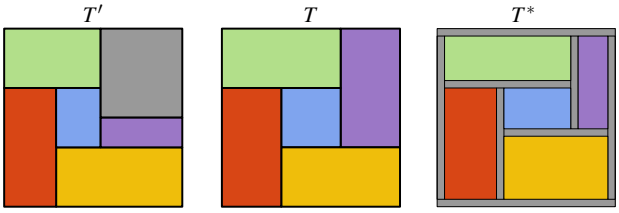


Figure 3.2: Treemaps T' (with gray rectangle inserted), T, and T^* (with gray area spread over maximal segments).

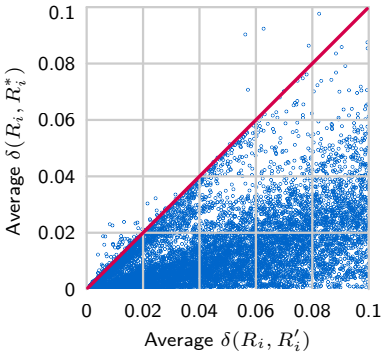


Figure 3.3: Scatter plot of the average layout change between T and T' or T^* for a random 25% sample of all algorithms and datasets.

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 δ 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

$$\sigma(R_i) = \max(0, \delta(R_i, R_i') - \delta(R_i, R_i^*)) \tag{3.3}$$

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.

3.4 DATA

The visual quality and/or stability of treemaps clearly depends 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 of different treemapping algorithms for each class.

3.4.1 *Data features*

Our methodology is inspired by the framework proposed by Smith-Miles et al. (2014) 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 (Espadoto et al., 2019) or in machine learning (Bishop, 2006). Based on the discussion of treemapping algorithms in Section 3.2, 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-dependent) trees, such as the minimum, maximum, and average node degrees, the (im)balance of the tree structure (Boorman and Oliviera, 1973; Kuhner and Yamato, 2015), and 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 with certainty a discriminating factor between algorithms, see our discussion in Section 3.2. 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 data feature than the total number of nodes. Nevertheless, we decided not to include the branching factor in our experiments, for the following two reasons. First of all, 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 of the algorithms do no give an indication that the branching factor is able to predict if an algorithm A will perform better than an algorithms 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 3.4.2), 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.

3.4.2 *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 tree datasets.

Levels of hierarchy (3 subclasses). We use three ranges for classification: 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 geo-spatial 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 (Vliegen et al., 2006; Bruls et al., 2000). 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 (Vliegen et al., 2006). 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 (Hahn et al., 2014, 2017; Vernier et al., 2018b).

Variance of node weights (2 subclasses). We distinguish between low variance (LWV) and high variance (HWV). To ensure that the total number of tree nodes does not strongly influence our classification,

we use the coefficient of variation σ/μ to determine the subclass. The standard deviation σ and the mean μ 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 (3 subclasses). We distinguish between 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 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 μ and the standard deviation σ. 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 (3 subclasses). We distinguish between 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 (μ and σ) 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 3.5, we evaluate how the performance of treemapping algorithms depends on the classes, that is, if the classification is sensible.

3.4.3 *Datasets*

We collected a total of 2405 time-dependent 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 3.4 for the distribution of datasets over classes: clearly not all classes arise with equal frequency in our data sources.

WORLD BANK (URL, ACCESSED 04-07-2018): (2142 datasets) World development indicators such as agriculture, rural and urban development, education, trade and health. Hierarchy

		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 3.4: Distribution of datasets over classes.

either according to the World Bank regional classification, grouping countries into subregions and continents, or no hierarchy present.

GITHUB (URL, ACCESSED 16-07-2018): (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 (URL, accessed 15-02-2017b).

MOVIES: (107 datasets) Movies from MovieLens (Harper and Konstan, 2016) and *TMDB* (URL, accessed 10-02-2018). We constructed a time-dependent hierarchy using the group-rows-by-attribute-value partitioning method (Telea, 2006; Vliegen et al., 2006). The hierarchy groups movies based on their 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* (URL, accessed 30-05-2016) contains the frequency of popular baby names in the Netherlands per year; *UN Comtrade Coffee* (URL, accessed 15-02-2017a) 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 (URL, accessed 03-07-2018a); and *Earthquakes* contains the time, location, depth and intensity of seismic phenomena provided by the USGS Earthquake Hazards Program (URL, accessed 03-07-2018b).

Importantly, note that the above 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 $\mathcal{F}$. The distributions of

data classes covered by our different data sources can be found in the supplementary material. The large and varied collection of World Bank datasets is able to cover all data classes with at most 3 levels of hierarchy (to which it is inherently limited). The GitHub and Movies datasets further cover a number of data classes with 4+ levels of hierarchy.

3.5 EXPERIMENTAL RESULTS

We ran all 14 algorithms on all time steps of all 2405 datasets, generated the baselines for all these instances (Section 3.3), and recorded all layouts, that is, the positions of all rectangles $R_i(t)$ at all time steps t. Per dataset we aggregate our results for all metrics and algorithms by first by taking the mean over all rectangles in a single time step, and then by taking the mean again over all time steps. This is necessary since the number of rectangles may differ per time step.

We focus on two specific questions: We first explore the *validity* of our data classification (Section 3.5.1) and then we study the *performance* of all algorithms with respect to visual quality and stability across varying data features (Section 3.5.2). In the supplementary material we additionally compare the performance of all algorithms on each data *class* separately. We believe that the resulting visual summary will help researchers and practitioners choose a suitable treemapping algorithm for their data.

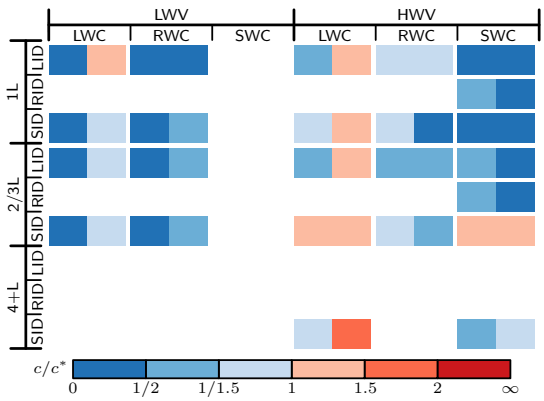


Figure 3.5: For each data class with at least 50 datasets, the ratio of the consistency score (visual quality on the left, stability on the right) between the data class and the baseline.

3.5.1 *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

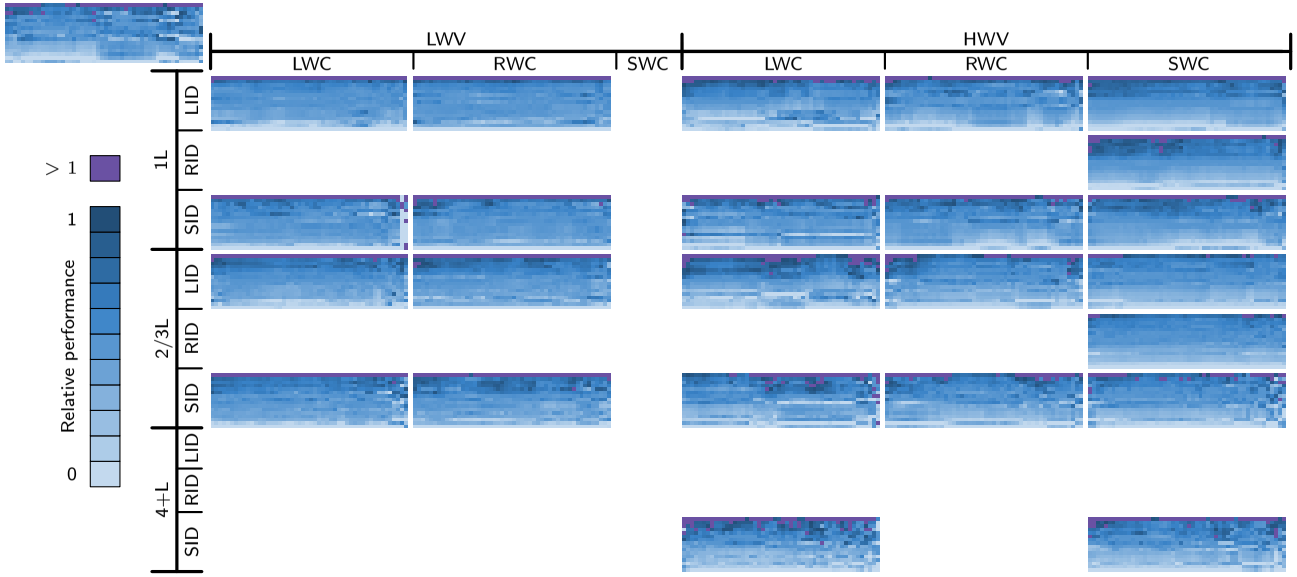


Figure 3.6: Visual quality: matrix plots for each data class with at least 50 datasets plus baseline (left top). In each matrix plot, rows correspond to algorithms, columns to datasets. The lighter the color, the better the relative performance, capped at 1 (purple).

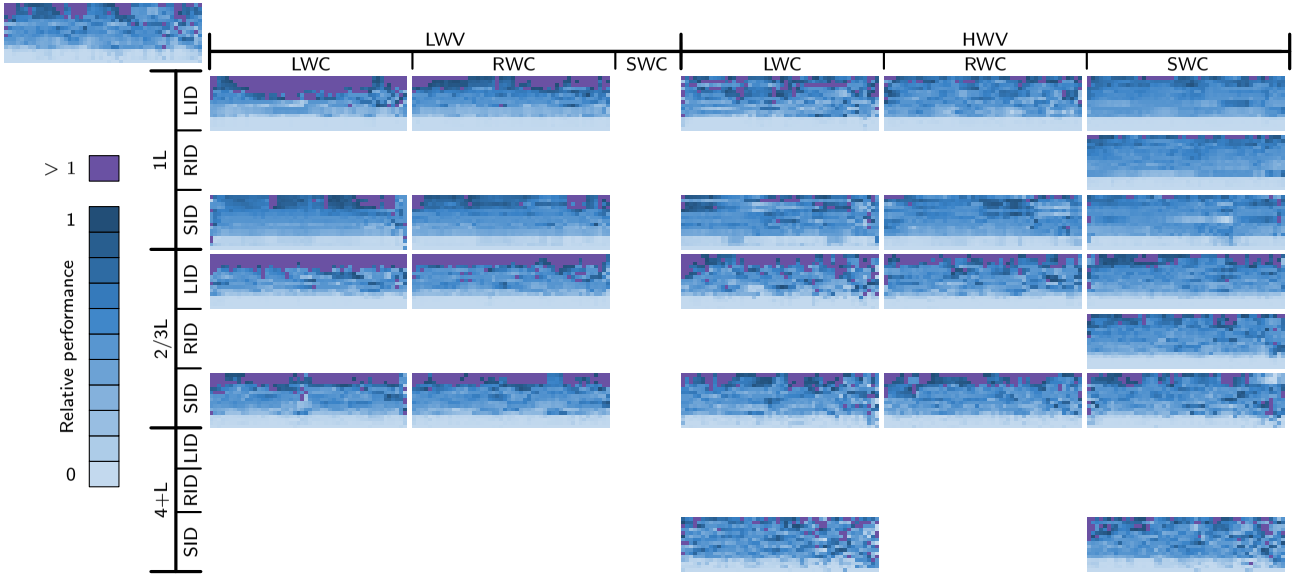


Figure 3.7: Stability: matrix plots each data class with at least 50 datasets plus baseline (left top). In each matrix plot, rows correspond to algorithms, columns to datasets. The lighter the color, the better the relative performance, capped at 1 (purple).

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 different 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 we sum up these variances over all algorithms. This results in a *consistency score* c for a collection of datasets. Figure 3.5 displays the consistency scores (for visual quality 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 is smaller than c^*; a cell is colored red (less consistent) if c is larger than 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 3.2, 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 3.6 and 3.7). 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 3.6 shows the matrix plots for visual quality, with the corresponding matrix plot for the baseline collection at the left-top. Figure 3.7 shows the matrix plots for stability.

First consider the matrix plot for visual quality (Figure 3.6). 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 3.5. 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 3.7. 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 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 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.

3.5.2 *Performance analysis across features*

The analysis in Section 3.5.1 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 its 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 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 3.8,3.9,3.10,3.11. Each point in each figure represents the score for one algorithm on one subclass of the feature, for example, low weight variance. We draw a polyline that connects the points of one algorithm and use glyphs to indicate the different subclasses. 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 (ρ) of 0.5 for a single rectangle indicates a 2-by-1 rectangle. A ρ of 0.25 however is perceptually much worse than a ρ of 0.5 in terms of area perception as can be inferred from Kong et al. (2010), coming close to their "extreme aspect ratios" of 4.5.

Levels of hierarchy. Figure 3.8 considers the levels of hierarchy feature, which has three values: 1L, 2/3L, and 4+L. From Figure 3.8, we see 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 3.9 considers the weight variance feature, which has two values: LWV and HWV. 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 3.10 considers the weight change feature, which has 3 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 tradeoff between stability and visual quality (see the slightly sloping line).

Insertions and deletions. Finally, Fig. 3.11 considers the insertions and deletions feature, which has three values: LID, RID, and SID. The plot shows a similar variation of visual quality and stability as seen for

the weight change feature (Fig. 3.10). 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, 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.

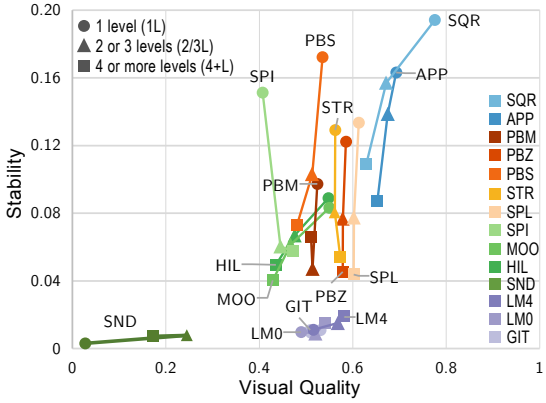


Figure 3.8: Visual quality *vs* stability as function of the levels of hierarchy feature.

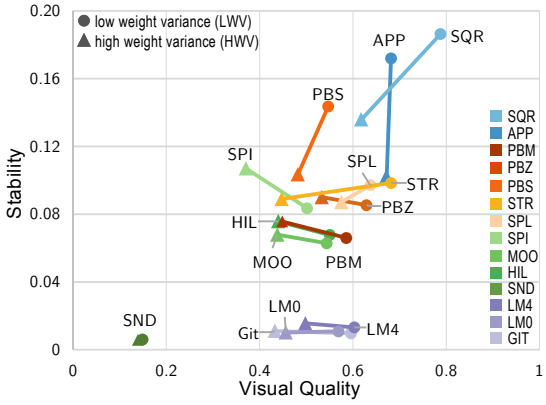


Figure 3.9: Visual quality *vs* stability as function of the variance of node weights feature.

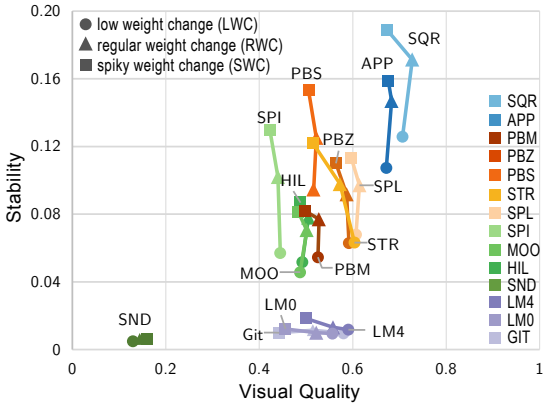


Figure 3.10: Visual quality *vs* stability as function of the weight change feature.

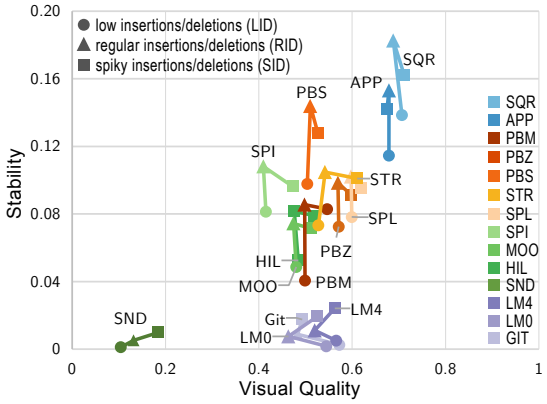


Figure 3.11: Visual quality *vs* stability as function of the insertions and deletions feature.

3.5.3 *Comparison of data classes*

We next compare the relative performance of all algorithms separately on all data classes. Figs. 3.12 to 3.14 support 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 3.5.2. Empty cells indicate data classes for which we did not find datasets. Figs. 3.12 to 3.14 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 performance on 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 indicate algorithms which score similarly regardless of data class.

There are a number of additional insights we can obtain from Figs. 3.12 to 3.14. When we consider only the visual quality, we see that 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 for the other depth subclasses it depends on the exact data class. If only the stability is important, SND almost always scores best regardless of the data class, 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. When considering which algorithm is best for both stability and visual quality, there are no easy answers. There is no algorithm that performs best on both in any of the data classes and hence the answer depends on the desired trade-off and the data class in question.

3.6 DISCUSSION AND CONCLUSION

We performed an extensive quantitative evaluation of rectangular treemaps for time-dependent data. To do so, we introduced a new methodology based on baseline treemaps to measure the stability of time-dependent 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-dependent data sets 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 var-

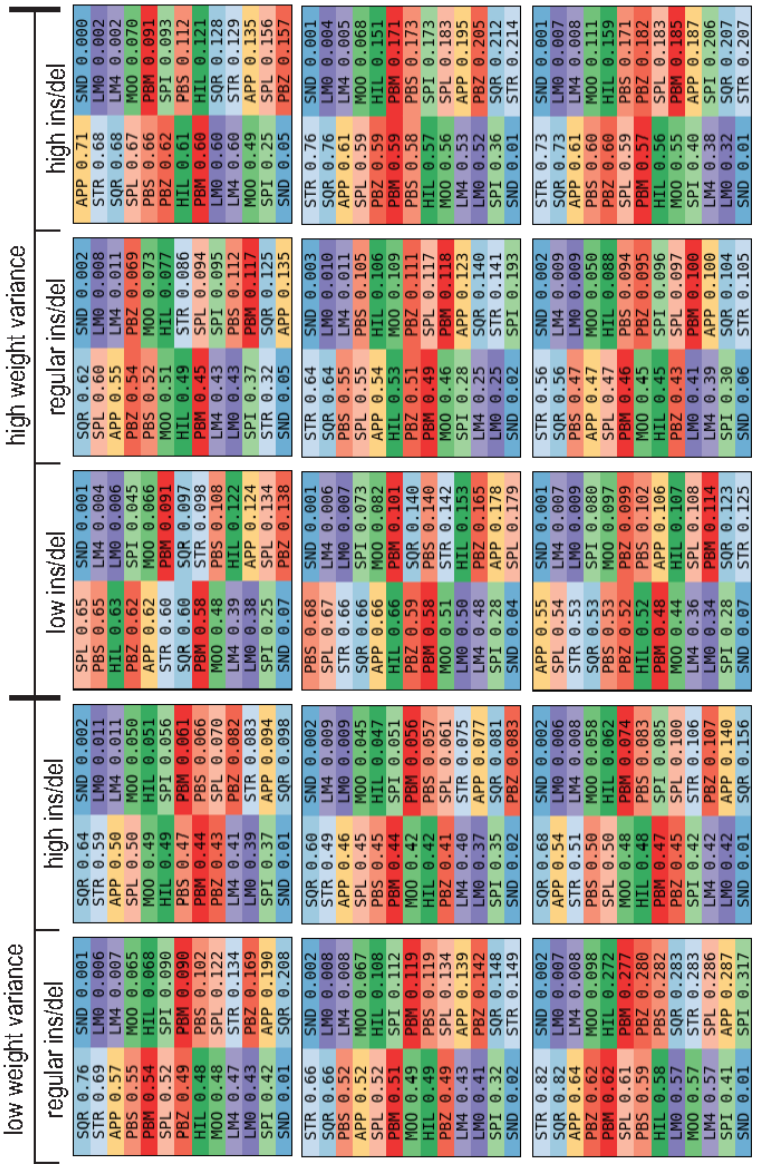


Figure 3.12: Relative ranking of treemapping algorithms for all data classes. Each table cell shows algorithms in top-down decreasing order of average visual quality (left column) and average stability (right column).

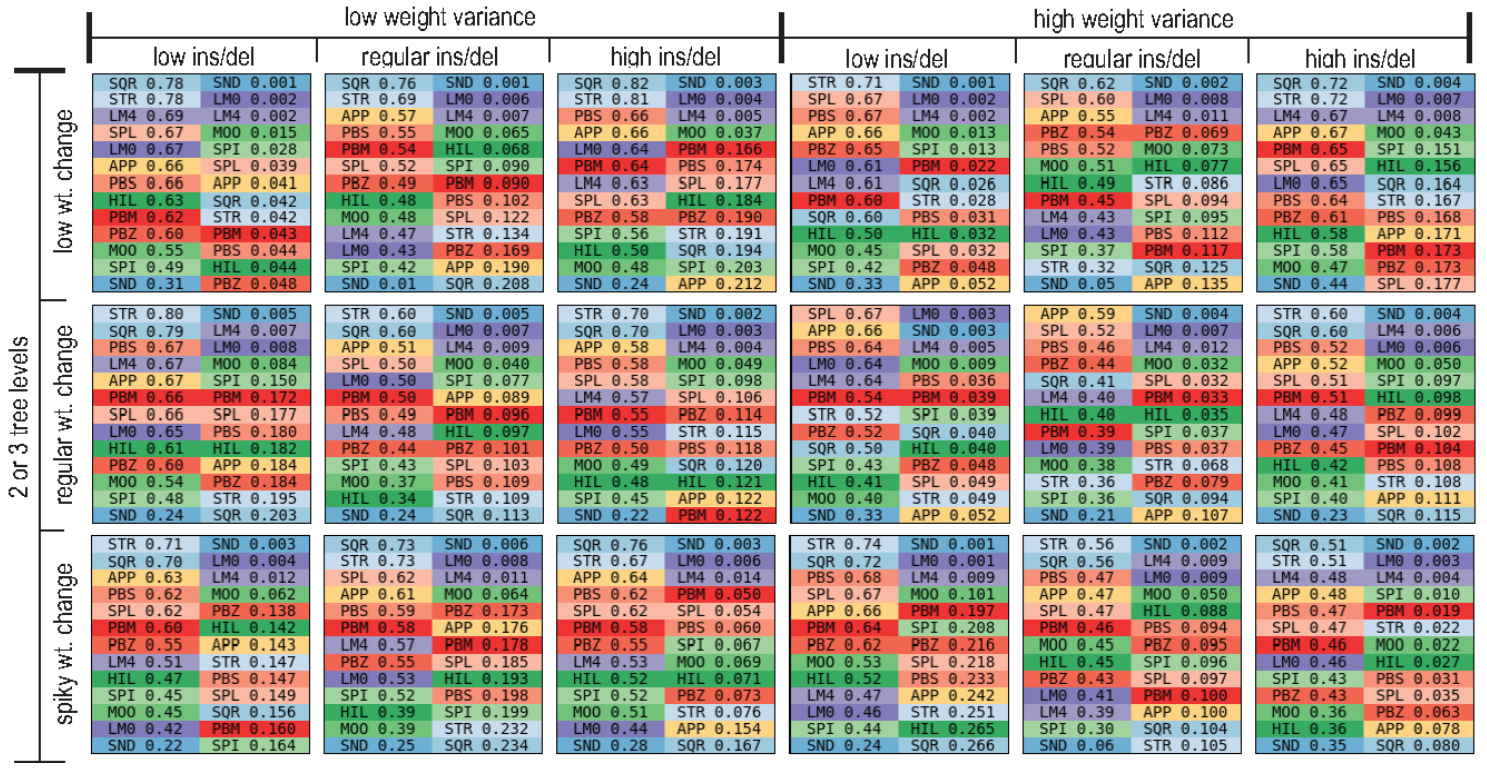


Figure 3.13: Relative ranking of treemapping algorithms for all data classes. Each table cell shows algorithms in top-down decreasing order of average visual quality (left column) and average stability (right column).

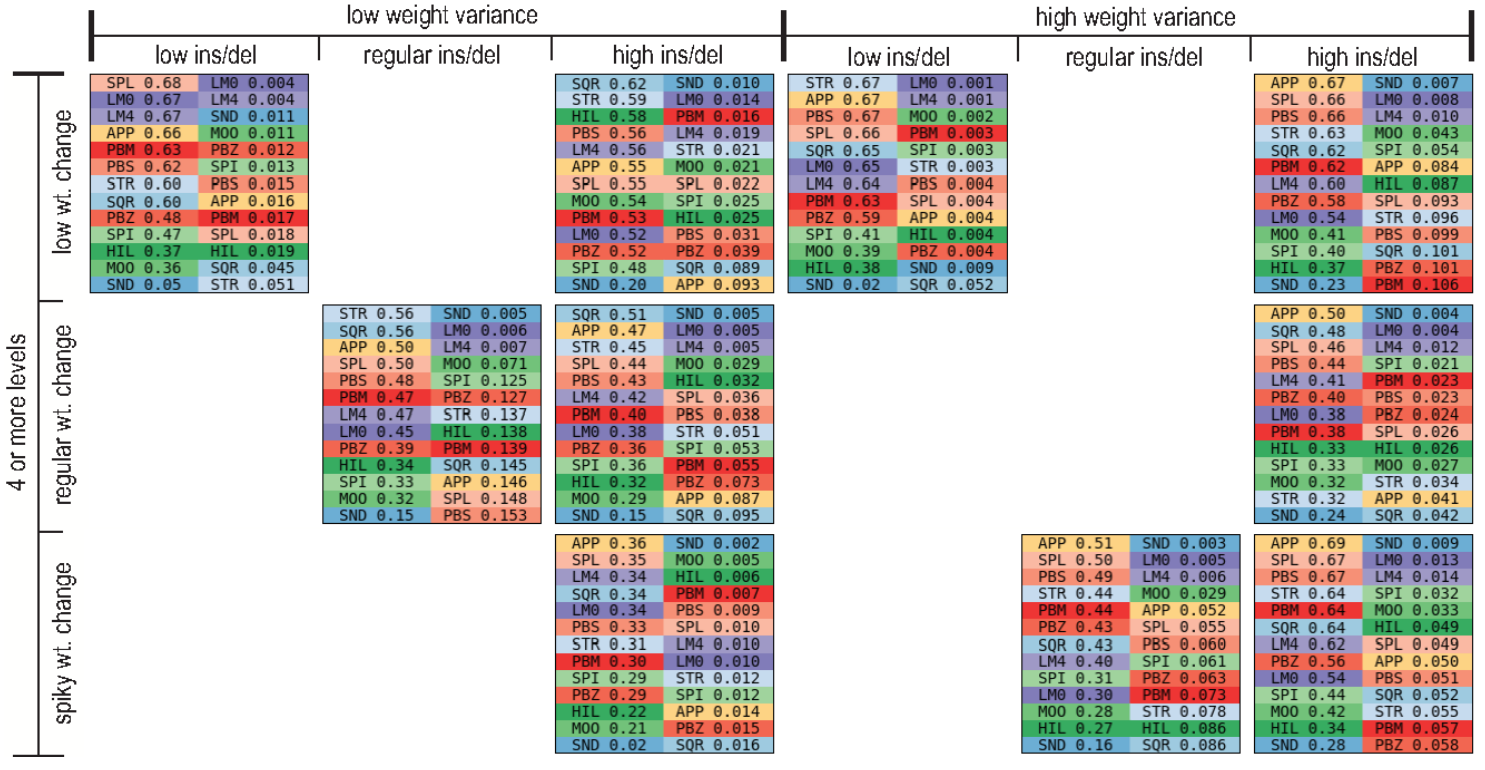


Figure 3.14: Relative ranking of treemapping algorithms for all data classes. Each table cell shows algorithms in top-down decreasing order of average visual quality (left column) and average stability (right column).

ious 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 (The Authors, 2020a).

Limitations and future work. Our experiments show that our identified features 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: data sets 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-dependent 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.

To complement our quantitative evaluation it would naturally be of interest to evaluate the performance of treemapping algorithms in various usage scenarios through user studies. The two metrics we use for

visual quality and stability are both perceptually salient according to studies performed in previous work. However, a study that evaluates the combination of and the trade-offs between visual quality and stability could deliver important insights as to where on the Pareto-front an optimal treemapping algorithm should lie.

Finally, our evaluation currently does not measure the run-time and correspondingly the computational scalability of the algorithms used in our experiments. Our implementations are not (equally) optimized and hence a fair comparison is currently risky if not 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.

4 IMPROVED TREEMAPPING FOR DYNAMIC DATA

The previous two chapters have shown two evaluations of treemapping algorithms used for visualizing time-dependent hierarchies. The first focusing only on datasets obtained from evolving software systems (Chapter 2) and the second being done for a much wider spectrum, and number, of datasets obtained from many application domains (Chapter 3). We finalize the treemap track of this thesis by proposing a new algorithm called Greedy Insertion Treemap. *GIT is a simple and scalable technique that achieves a good balance between temporal stability and visual quality, scoring better results than existing methods, as witnessed by the comprehensive benchmark introduced in Chapter 3.*

Abstract: Computing treemap layouts for time-dependent (dynamic) trees is an open problem in information visualization. In particular, the constraints of spatial quality (cell aspect ratio) and stability (small treemap changes mandated by given tree-data changes) are hard to satisfy simultaneously. Most existing treemap methods focus on spatial quality, but are not inherently designed to address stability. We propose here a new treemapping method that aims to jointly optimize both these constraints. Our method is simple to implement, generic (handles any types of dynamic hierarchies), and fast. We compare our method with 14 state of the art treemaping algorithms using four quality metrics, over 28 dynamic hierarchies extracted from evolving software codebases. The comparison shows that our proposal jointly optimizes spatial quality and stability better than existing methods.

4.1 INTRODUCTION

Understanding the evolution of large and long-lasting software projects is a major aspect of program comprehension. Typically, evolution data for such projects is mined by fact extraction tools from existing software control management systems handing software repositories, such as Git (GIT, 2018), Subversion (Subversion, 2018), and CVS (CSV, 2018). Several types of data attributes are collected (and explored) in this way, including the identity of software items of interest (*e.g.*, packages, folders, files, classes, methods), various quality attributes measured on them (*e.g.*, testability, maintainability, modularity, and readability metrics (Lanza and Marinescu, 2006)), and relations that interrelate these

This chapter is based on the paper "A Stable Greedy Insertion Treemap Algorithm for Software Evolution Visualization" (Vernier et al., 2018a)

items. *Hierarchy* relations, which describe the containment or aggregation of software items, play a central role in virtually all such evolution analyses, since they offer a powerful and natural way to examine the (typically large) evolution data at multiple levels of detail. As such, methods that can depict time-dependent hierarchies are a central element of the program evolution toolset.

Dynamic, or time-dependent, treemaps are one of the most effective techniques for displaying time-dependent hierarchies. Compared to other techniques, such as node-link tree layouts, they use basically every pixel of the available screen space to display information, and as such scale to tens of thousands of items (tree nodes) per time step. Many treemap methods exist for handling static (time-independent) hierarchies (Shneiderman, 1992; Bruls et al., 2000), which also have been shown to optimize various quality measures that help readability, such as aspect ratio (Bruls et al., 2000; Nagamochi and Abe, 2007) and relative positions of nodes (Wood and Dykes, 2008; Shneiderman and Wattenberg, 2001; Ghoniem et al., 2015; Buchin et al., 2011; Duarte et al., 2014). However, far fewer methods are available for dynamic trees (Hahn et al., 2014; Sondag et al., 2017; Hahn and Döllner, 2017). One key problem for dynamic treemapping is *instability*, *i.e.*, the fact that relatively small changes in a tree can induce disproportionately large changes in the resulting treemaps. Finding a good way to quantify and reduce instability is an open problem for dynamic treemap algorithms.

In this chapter, we address the above limitations with two main contributions. Firstly, we propose a new dynamic treemap algorithm, called Greedy Insertion Treemap (GIT). GIT aims to preserve treemap-cell neighborhoods over time by constructing an initial so-called Layout Tree (LT), which is next incrementally updated as the tree data changes, so as to minimize undesired treemap-layout changes. Secondly, we evaluate the quality of GIT both in the spatial domain and the temporal domain against a large set of well-known treemap algorithms using several established quality metrics, and on a large set of dynamic hierarchies extracted from real-world software repositories. Our evaluation results show that GIT strikes a better balance between spatial and temporal quality than the existing competing methods we evaluated against. As GIT has a simple and computationally scalable implementation, we argue that it represents a valuable contribution to the toolset of techniques needed by program evolution comprehension.

The structure of this chapter is as follows. Section 4.2 outlines existing work on (dynamic) treemapping and related quality metrics, and their use in program evolution comprehension, and also introduces the treemap methods we compare against. Section 4.3 details our new GIT algorithm. Section 4.4 presents our evaluation methodology for GIT and the obtained results are revealed in Section 4.5 . Section 4.6 discusses our proposal and outlines directions for future improvement.

4.2 RELATED WORK

In this section, we will discuss the Algorithms (Section 4.2.1) and Quality Metrics (Section 4.2.2) present in the dynamic treemap literature. Let $T = \{n_i\}$ be a hierarchy (tree) with nodes n_i, each having a weight value $a_i \geq 0$. Weights are given for leaf nodes and computed for non-leaf nodes as the sum of their children weights, respectively. Let $\mathcal{T}(T)$ be the treemap layout of T, with a rectangle cell r_i assigned to each n_i, so that the area of r_i equals a_i.

4.2.1 *Algorithms*

Time-dependent hierarchies $T(t)$ are a central artifact to explore in program evolution comprehension. Since such analyses usually involve tens or even hundreds of time steps t, small-multiple visualizations (one image per time step) do not scale well, hence showing an animated layout of the changing hierarchy is preferred (Diehl, 2007). For this, several techniques construct a so-called union tree $\cup_t T(t)$, build a single layout of this union tree, display it using Icicle plots (Kruskal and Landwehr, 1983) or Sunburst diagrams (Clark, 2006), and then highlight changes of $T(t)$ over time in it (Hurter et al., 2013). While this approach minimizes instability (layout changes) over time, and is simple to implement, it cannot handle long time sequences and/or large trees.

Treemaps cope well with the need for handling large trees (Schulz et al., 2011; Shneiderman and Plaisant, 2017; Schulz, 2011; von Landesberger et al., 2011). Slice and dice (SND) treemaps introduced the idea but were found to create poor aspect-ratio (AR) cells which are hard to see (Shneiderman, 1992). Squarified treemaps (SQR) propose a heuristic that yields good (close to one) AR values (Bruls et al., 2000). A subsequent algorithm (APP) was designed to approximate the optimal AR (Nagamochi and Abe, 2007). While treemaps were originally designed to handle time-independent trees, the need for *stability* was soon revealed – that is, small changes in the input tree T should yield only small changes in the treemap $\mathcal{T}(T)$. Several algorithms were designed to improve stability. Ordered treemaps (OT) (Shneiderman and Wattenberg, 2001) and Strip treemaps (STR) (Bederson et al., 2002) lay out cells r_i using a given order of the nodes of T, using different heuristics – Pivot-by-Middle (PBM), Pivot-by-Size (PBZ), and Pivot-By-Split-Size (PBS) (Shneiderman and Wattenberg, 2001). Other algorithms lay out cells along a space-filling fractal-like curve, *e.g.*, Spiral (SPI) (Tu and Shen, 2007), and Hilbert (HIL) and Moore (MOO) methods (Tak and Cockburn, 2013). Yet another ordering technique considers node similarities: Spatially-Ordered Treemaps (SOT) (Wood and Dykes, 2008) processes sibling nodes ordered by decreasing similarity; NMap (Duarte et al., 2014) places cells according to the similarity of their nodes using dimensionality reduction. Variants thereof include NMap Alternate

Cuts (NAC), which splits the screen space alternating horizontal and vertical slices; and NMap Equal Weights (NEW) which aims to create similar-size cells.

Stability becomes a major concern when treemapping time-independent trees with potentially long evolution and large variations. However, only a few methods explicitly aim to treat dynamic data. Stable treemaps (Sondag et al., 2017) aim to improve both AR and stability by using non-sliceable layouts. However, this method is computationally expensive and not trivial to implement. Voronoi treemaps (Balzer and Deussen, 2005; Balzer et al., 2005) achieve, in general, good AR values, and have been adapted to also handle dynamic trees to visualize software structure evolution (van Hees and Hage, 2017; Gotz, 2011). There exist also methods that propose other cell shapes, or combinations of multiple shapes, such as bubble treemaps (Görtler et al., 2018), jigsaw treemaps (Wattenberg, 2005), and orthoconvex treemaps (Berg et al., 2014). However, such methods have not been specifically designed with the aim of maximizing stability.

4.2.2 *Metrics*

As outlined in Sec. 4.2.1, treemap quality consists of two main components:

Spatial quality captures how readable the treemap geometry is. The best known, and most used, metric for this is the aspect ratio (AR) of the cells r_i which should ideally reach one. The so-called readability metric measures how often a user's gaze changes direction while reading an ordered treemap along the predefined node ordering (Bederson et al., 2002). The continuity metric measures how often cells of nodes which are close in the given node ordering are far apart in the treemap (Tu and Shen, 2007).

Stability metrics capture how easily can a user understand the changing geometry of a dynamic treemap. This is measured essentially by quantifying the visual change $\delta(r_i(t), r_i(t+1))$ of the cells r_i, and then aggregating such visual changes into a single value using some function S. Early on, Shneiderman and Wattenberg (2001) defined the Layout Distance Change metric, where they used for δ the distance between the vectors $(x_i(t), y_i(t), w_i(t), h_i(t))$ and $(x_i(t+1), y_i(t+1), w_i(t+1), h_i(t+1))$, x and y being the coordinates of the top-left corner, and w and h, the width, and the height of a rectangle r_i. They defined S as the average of δ for all cells and revisions. Later, Hahn et al. (2014) use for δ the distance between the centers of $r_i(t)$ and $r_i(t+1)$ and also average for S. Tak and Cockburn (2013) use for S the variance and define δ as (Shneiderman and Wattenberg, 2001). They also propose a drift metric, which measures how much a cell's center moves away from its average position over long time intervals. Recently, we have seen new metrics that measure stability not by looking only at a single cell's position relative

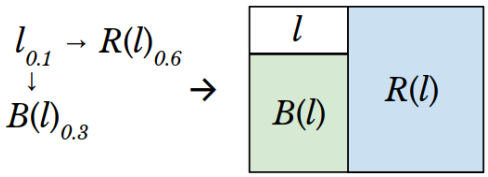


Figure 4.1: Space partitioning from LT.

to its past states, but take into consideration the relationships between all cells in the layout. Hahn et al. (2017) propose the relative direction change, which measures angle differences between all centroids in the layout between consecutive time-steps, and Sondag et al. (2017) propose similar metric, where δ measures how a cell moves with respect to all its neighbors, where S is again the average. We will discuss these metrics further, and also propose a new one in Section 4.4.1.

4.3 GREEDY INSERTION TREEMAP

As outlined in Sec. 4.2, many treemapping methods exist in the literature, and these have been evaluated by several metrics for both spatial quality and stability. However, examining the above in more detail, we find two limitations: (a) most existing treemap methods have been designed without the *explicit* aim of maximizing stability; (b) among the few methods where stability was an aim, there is no clear optimal method which yields both good spatial quality and stability for long time sequences of trees exhibiting a high dynamics in terms of node additions, deletions, and weight changes. We next propose a method, Greedy Insertion Treemap (GIT), that aims to outperform the current state-of-the-art in these two respects.

GIT is designed from the start with the aim of increased stability. For this, GIT aims to preserve cell neighborhoods in the treemap over time. To this end, we use a so-called Layout Tree (LT) help data structure (not to be confused with the tree T we want to visualize). Each node $l \in LT$ represents a treemap cell, and may have two subtrees: (a) $R(l)$ is rooted at the top-right corner of l; and (b) $B(l)$ is rooted at the bottom-left corner of l. Together with the cell weights, LT fully encodes a treemap $\mathcal{T}$. Indeed, we can construct $\mathcal{T}$ by traversing LT breadth-first. During this, for each $l \in LT$, we compute the total weight of its subtrees $R(l)$ and $B(l)$, and cut the remaining drawing space vertically and horizontally according to these summed weights, as illustrated in Fig. 4.1.

GIT proceeds in two phases: initialization and update, as follows.

Initialization: To start with, we need to construct LT from the first tree $T(t = 0)$ in our sequence. For this, we can use basically any method $\mathcal{T}_{init}$ that constructs a (static) treemap for $T(0)$ from which we can next gen-

erate LT with the properties (a) and (b) mentioned above. We have experimented with two such initialization methods. First, we constructed LT from a squarified treemap (*i.e.* $\mathcal{T}_{init}$ is the SQR algorithm), since SQR is well known to yield very good AR values. Alternatively, we propose a simple heuristic $\mathcal{T}_{init}^{direct}$ that directly builds LT from the initial tree $T(0)$. Both initialization methods are compared next in Sec. 4.5.

To explain our heuristic $\mathcal{T}_{init}^{direct}$, consider a single-level tree $T = [(n_i, a_i)] = [(A, 10), (B, 2), (C, 8), (D, 4), (E, 1), (F, 3)]$, to be laid our, for simplicity, in a square drawing area S of size 1; handling general trees is trivial by top-down recursion. We build LT by sequentially adding each $n_i \in T$ to it (Fig. 4.2). After each addition, we rebuild $\mathcal{T}$ from the current LT as explained above, so it covers the entire S. Thus, existing nodes are 'squeezed' to make space for the new nodes. In our example, we first add node A, which will cover the entire S. To add B, we find the node $n \in LT$ having the worst aspect-ratio cell $c \in \mathcal{T}$. If $w_c \geq h_c$, we add B directly right of n (as in our example), else we add B directly below n, and update $\mathcal{T}$ from the new LT again. For the third node C, as the worst-aspect-ratio cell is B, and since $h_B > w_B$, we add C below B and update $\mathcal{T}$ from LT again. Fig. 4.2(d-f) shows the addition of the remaining nodes of T.

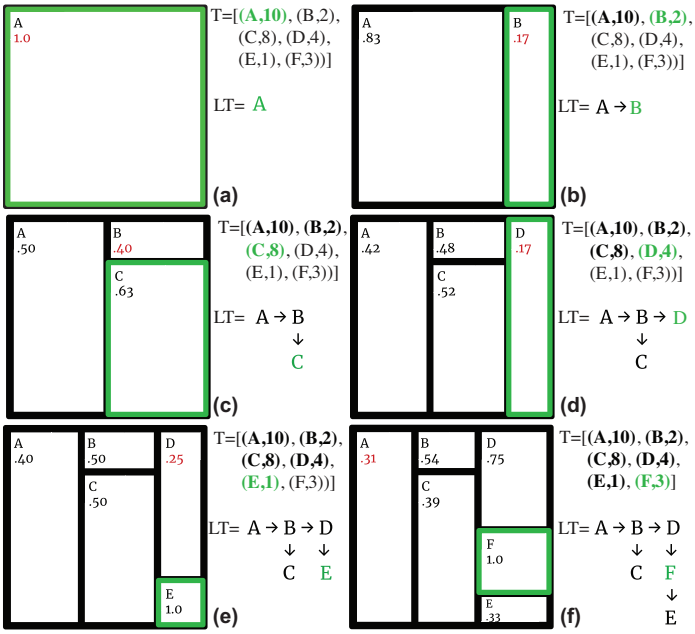


Figure 4.2: Building the initial layout tree LT (green: inserted cells).

For didactic purposes, nodes were added in alphabetical order, but in reality, we want nodes to be added in random order, hence when

dealing with truly hierarchical data, one sub-tree is not completely laid out before its siblings, which could cause it to be 'squeezed'.

Update: We now have an initial treemap $\mathcal{T}$ and its LT. We next edit LT to handle weight changes, node additions, and node deletions as T changes. *Weight* changes do not change LT. *Additions* are handled just as adding regular nodes when building the initial LT using $\mathcal{T}_{init}^{direct}$. Additions tend to increase the cells' aspect ratios, so we do them after node removals and weight changes. *Removals* are done by editing LT as follows (see also Fig. 4.3):

1. If a node $n \in LT$ has a $B(n)$ subtree, we replace n by $B(n)$;
2. else if n has a $R(n)$ subtree, we replace n by $R(n)$;
3. else n has no subtrees, so we just remove it from LT.

After handling all changes in a new revision of T, we rebuild $\mathcal{T}$ from LT, as already explained. As we show next in Sec. 4.5, GIT scores a very good balance of spatial quality *vs* stability.

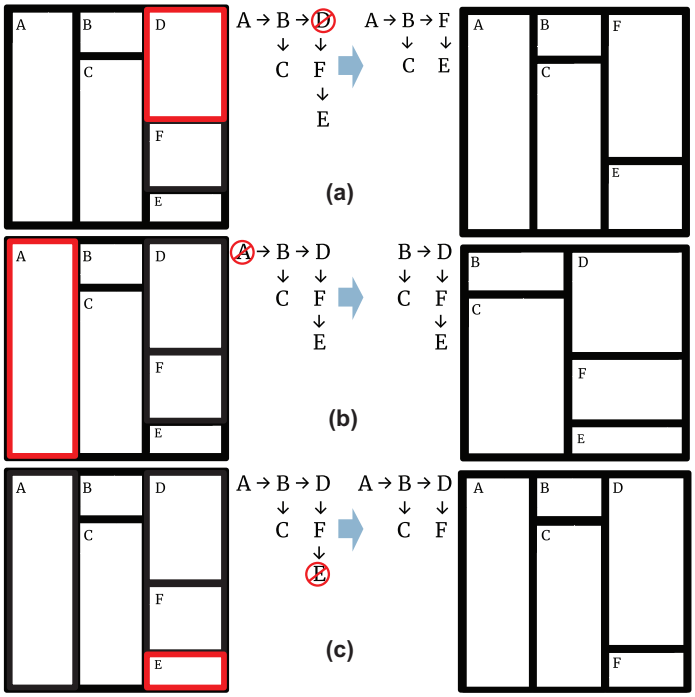


Figure 4.3: Removal of nodes (red) from layout tree and its treemap.

4.4 EVALUATION

To evaluate GIT, we considered the following aspects:

4.4.1 *Metrics*

To evaluate the quality of GIT, we proceed as follows. For spatial quality, we use the well-known Aspect Ratio metric (*AR*) (Bruls et al., 2000). For each cell c in a treemap $\mathcal{T}$, $AR_c = min(w_c, h_c)/max(w_c, h_c)$. This metric is considered by virtually all rectangular treemap evaluations we are aware of.

For stability, we compute three metrics. The first two are the Shneiderman-Wattenberg's Layout Distance Change (*LDC*) (Shneiderman and Wattenberg, 2001) and Tak-Cockburn's Location Drift (*LD*) (Tak and Cockburn, 2013), already introduced in Sec. 4.2. The *LDC* metric captures the instability of a cell between consecutive revisions. In contrast, the *LD* metric captures the deviation of a cell's position over all timesteps.

We also propose a (new) third metric, which extends *LDC* to also consider the change of the *data*. We define the *visual change* of a cell c_i as the Euclidean distance traveled by the four corners of the rectangle r_i between t and $t+1$, normalized by the treemap diagonal $\sqrt{W^2 + H^2}$, so $\delta v_i \in [0, 1]$. Next, we define the *data change* of c_i as $\delta a_i = |a_i(t) - a_i(t+1)|$, where a_i is the relative weight of node c_i. With these, we define the stability Q_i of a cell c_i in a treemap as

$$Q_i = (1 - \delta v_i)/(1 - \delta a_i). \tag{4.1}$$

We define the stability Q of an entire treemap as the average of its cells' stabilities Q_i. In contrast to *LDC*, Q measures how much a rectangle changes *in relation to its data change*. Measuring only absolute changes of rectangles (*LDC*) does not, we believe, fully characterize stability. Indeed, a rectangle could (and should) change a lot if its underlying cell's weight changes a lot. However, this does not mean necessarily that the treemap algorithm is unstable.

4.4.2 *Techniques*

We tested GIT against 14 other treemapping algorithms: Approximate (APP), Hilbert (HIL), Stable treemaps (LM0, LM4), Moore (MOO), NMap-Alternate-Cuts (NAC), NMap-Equal-Weights (NEW), Pivot-by-Middle (PBM), Pivot-by-Size (PBZ), Pivot-by-Split-Size (PBS), Slice- and-Dice (SND), Spiral (SPI), Squarified (SQR), and Strip (STR). For NMap, we use as seed layout the one computed by SQR (Duarte et al., 2014). We did not consider non-rectangular treemap methods in the evaluation, since

not all the metrics in Sec. 4.2.2 directly generalize to non-rectangular cells.

4.4.3 *Datasets*

We extracted 28 dynamic hierarchies by mining the structure of software projects (folders, files, classes) from 28 corresponding public GitHub repositories, using a custom automated pipeline that scans all available revisions and extracts the code structure using Understand (SciTools, 2017). As weights w_i, we use the number of lines of code of the respective items. Other software quality metrics delivered by Understand can be used instead, if desired. For more details on this process, we refer to (da Silva et al., 2016). The considered repositories have quite different sizes, number of revisions, hierarchy depths and shapes, number of developers, and code type (programming languages and application types). Statistics about the datasets are available in Table 2.

4.5 RESULTS

We evaluate GIT on the aforementioned datasets, algorithms, and metrics collection from several perspectives, by answering a series of questions. Below, average stability S refers to the average of the *LDC* and Q metrics introduced in Sec. 4.2.2. All results that we were not able to fit in this chapter can be found at our online repository(The Authors, 2018).

4.5.1 *How does GIT's initialization affect its quality?*

As outlined in Sec. 4.3, we can initialize GIT with various treemap layouts, such as squarified (SQR) or using the direct initialization illustrated in Fig. 4.2. Intuitively, one would think that SQR initialization is to be preferred, since SQR is well known for its high *AR* values. To test this, we ran GIT using both initializations for all datasets. After initialization, the same regular GIT update mechanism is used in both cases. Figure 4.4 shows the per-dataset average stability and *AR* values. Interestingly, we see that the higher-*AR* SQR initialization actually yields slightly worse *AR* values for the entire sequence. For stability, the two initializations behave basically identically. We can explain this result by the fact that the GIT direct initialization follows the same heuristics as the update steps, while SQR forces GIT to start with a layout which needs more substantial updates next as the tree data changes. At a higher level, this experiment suggests that GIT performs very well using direct initialization. As such, we use this initialization in all subsequent experiments.

Dataset	Revisions	Nodes (total)	Average depth
animate.css	50	3454	2.87
AudioKit	22	11178	6.95
bdb	62	2658	3.83
beets	106	9844	3.75
brackets	88	120292	12.85
caffe	44	12969	4.93
calcuta	50	2882	10.76
cpython	321	584821	6.50
earthdata-search	46	18539	6.82
emcee	64	1746	3.62
exo	97	36436	11.88
fsharp	69	22906	7.89
gimp	72	170418	5.19
hospitalrun-frontend	38	16759	5.71
Hystrix	61	15530	13.29
iina	74	6849	4
jenkins	137	277185	11.94
Leaflet	84	13381	4.86
OptiKey	36	9782	6.72
osquery	37	14111	5.75
PhysicsJS	20	2022	4.6
pybuilder	53	5457	7
scikitlearn	88	48468	5.75
shellcheck	53	746	2.39
soundnode-app	35	3196	6.88
spacemacs	51	10201	4.96
standard	29	203	2
uws	122	4093	2.76
Totals:	2132	1458036	5.77

Table 2: Software evolution tree datasets used in the evaluation.

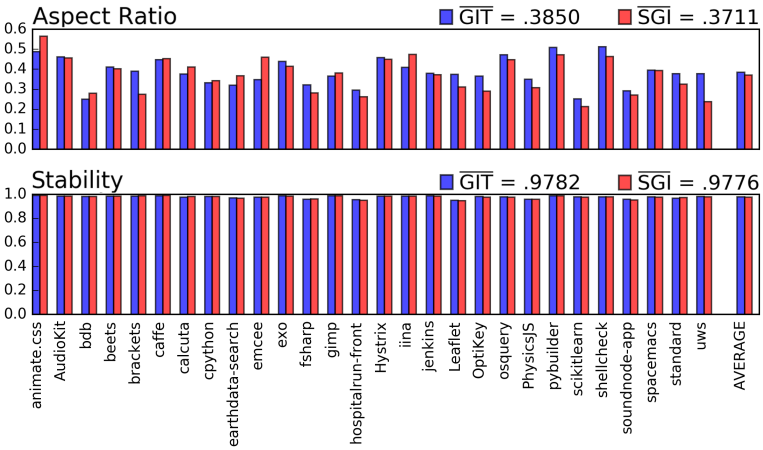


Figure 4.4: GIT performance using $\mathcal{T}_{init}^{direct}$ (GIT) *vs* squarified initialization (SGI).

4.5.2 *How do visual quality and stability vary over time?*

As we have already noted, spatial quality and stability are roughly inversely correlated desiderates – a treemap that scores well for one of these metrics tends to score less well for the other one. Hence, comparing how these metrics change in time is interesting. To answer this, we display, for one dataset and all tested algorithms, two charts showing the median (black), 25-75% range (green), and 5-95% range (gray) of the *AR* and *S* metrics (Fig. 4.5). We see that APP and SQR have the best *AR* values, and SND the worst *AR* values. The other algorithms, including GIT, score in-between. In contrast, GIT, LM0, and SND score the best for stability, while all other algorithms exhibit a non-negligible number of unstable time moments. This suggests that GIT strikes a good compromise between stability and aspect ratio.

While Fig. 4.5(right) shows how the per-timestep stability changes over time, it does not show us which actual instability patterns each method is prone to deliver. Knowing this is useful, as we can better understand what to expect in terms of (undesired) cell moves from a certain algorithm, including GIT. To show this, we plot the trails connecting all centers $k_i(t)$ of all rectangles $r_i(t)$ for consecutive t values over a given tree sequence (Fig. 4.6). We set the opacity of each line segment $(r_i(t), r_i(t+1))$ to the Euclidean distance $\|k_i(t) - k_i(t+1)\|$ normalized by the square root of the number of time steps. Hence, dark long lines show big moves (instability) while small moves (close to stability) are hardly visible. The image confirms the high stability of GIT – in contrast to most other methods, except SND, GIT creates smaller cell moves (shorter dark lines), and most of these are close to horizontal or

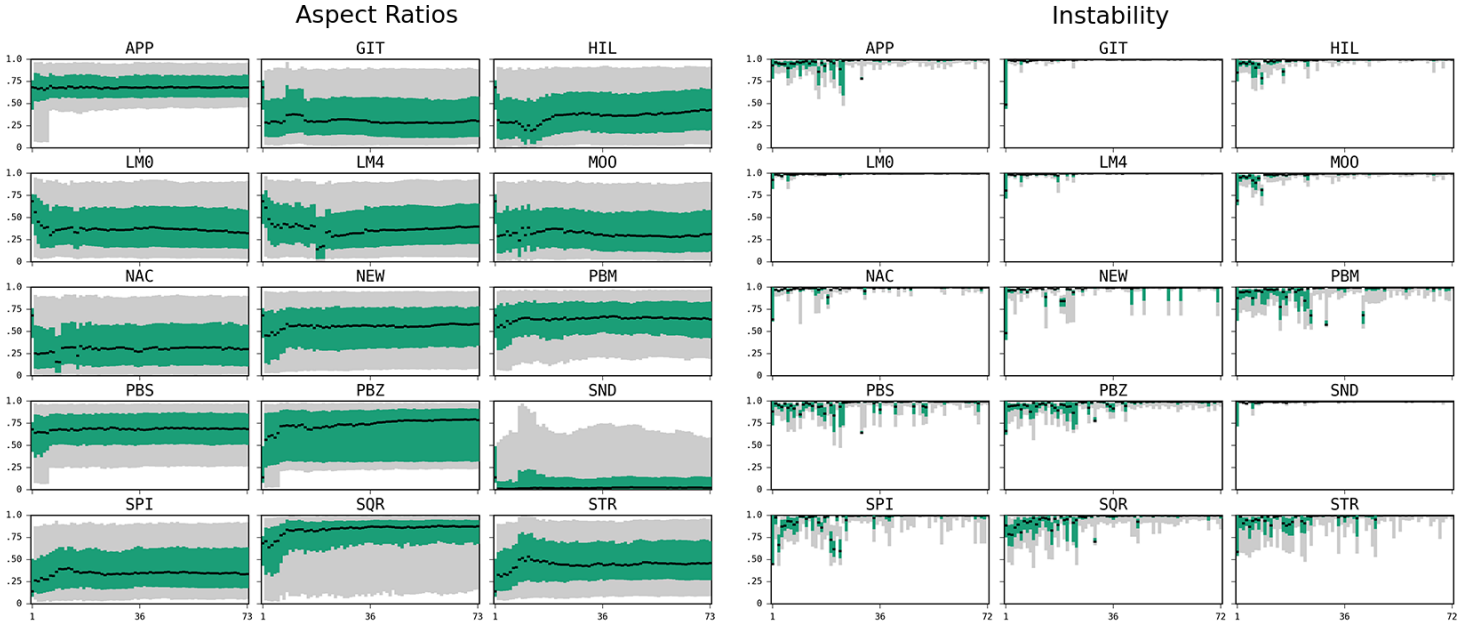


Figure 4.5: Distribution of aspect ratio (AR, $left$) and average stability (S, right) values over time for the GIMP dataset.

vertical. Interestingly, we see that other methods create quite different move patterns: SQR, PBS, PBZ, and PBM have mostly (large) diagonal moves. SPI shows a coil-like movement and in STR we see no vertical travel. Overall, we see that GIT is more stable not only because it yields smaller moves, but also because it constrains these to fewer motion directions, thus causes less complex dynamics (that the user must follow) in the resulting visualization. This can be also checked by watching the actual videos showing the algorithms in action (The Authors, 2018).

4.5.3 *How do all quality metrics vary over all datasets?*

The experiments so far do not show the individual stability metrics (including *LD*, which can be only computed for an entire sequence), nor, for space reasons, the metrics over all 28 tested tree sequences. To get more insight in how GIT performs in these respects, we show the per-dataset average values (for *AR* and the three stability metrics) for all tested methods, all datasets (Fig. 4.8). Cells are colored using a purple (low values) to yellow (high values) colormap. We observe the following: For *AR*, APP scores consistently better for most datasets than all other tested methods. SQR reaches the highest *AR* values, but only for a very few datasets. SND, as expected, scores overall the poorest. The remaining methods can be divided roughly into two groups, with NEW, PBM, PBS, STR, and PBZ scoring overall higher than GIT, HIL, LM0, LM4, MOO, and NAC. Concerning stability, SND scores consistently the best for all three considered metrics, and GIT, LM0, and LM4 come in the second place. This strengthens our earlier observation that GIT strikes a good balance between stability and spatial quality.

4.5.4 *How to summarize GIT's quality?*

As noted, GIT seems to strike a good balance between spatial quality and stability. We summarize both these metrics for GIT and all other algorithms using a star plot (Fig. 4.9). The figure shows a scatterplot with x mapping average stability S and y aspect ratio AR, respectively. Categorically colored points, one color per method, indicate the tested methods, attributed by their S and AR values over all datasets, all time steps. From each point (method), we draw lines connecting it with the S and AR values obtained for all the 28 tested datasets. A good algorithm has thus its 'star' center placed top-right and relatively short star arms, indicating consistent quality over the entire dataset collection. We see several patterns, as follows.

At a high level, stability is roughly inversely correlated with spatial quality – methods that score very well on one tend to score worse on the other. We see three groups of methods: APP, PBS, SQR, PBM, STR, PBZ and NEW score well on spatial quality, but poorly (except NEW) on

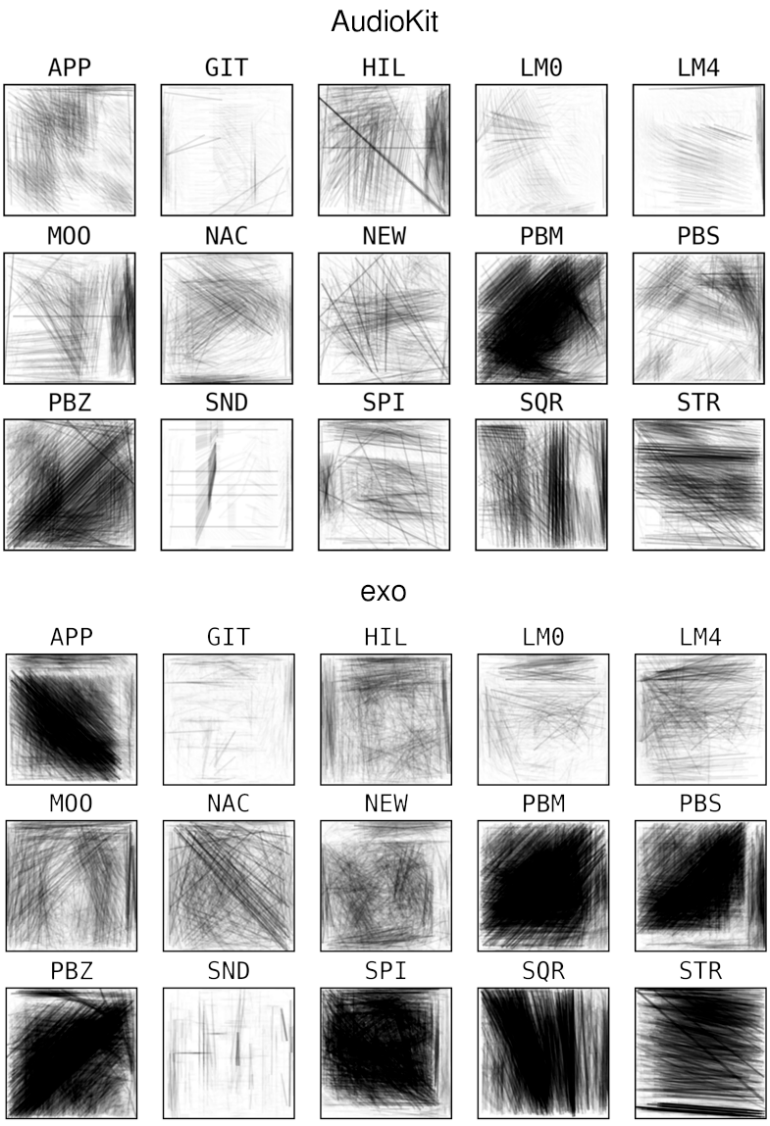


Figure 4.6: Instability (cell center motion) patterns, all methods, *AudioKit*, *exo* datasets.

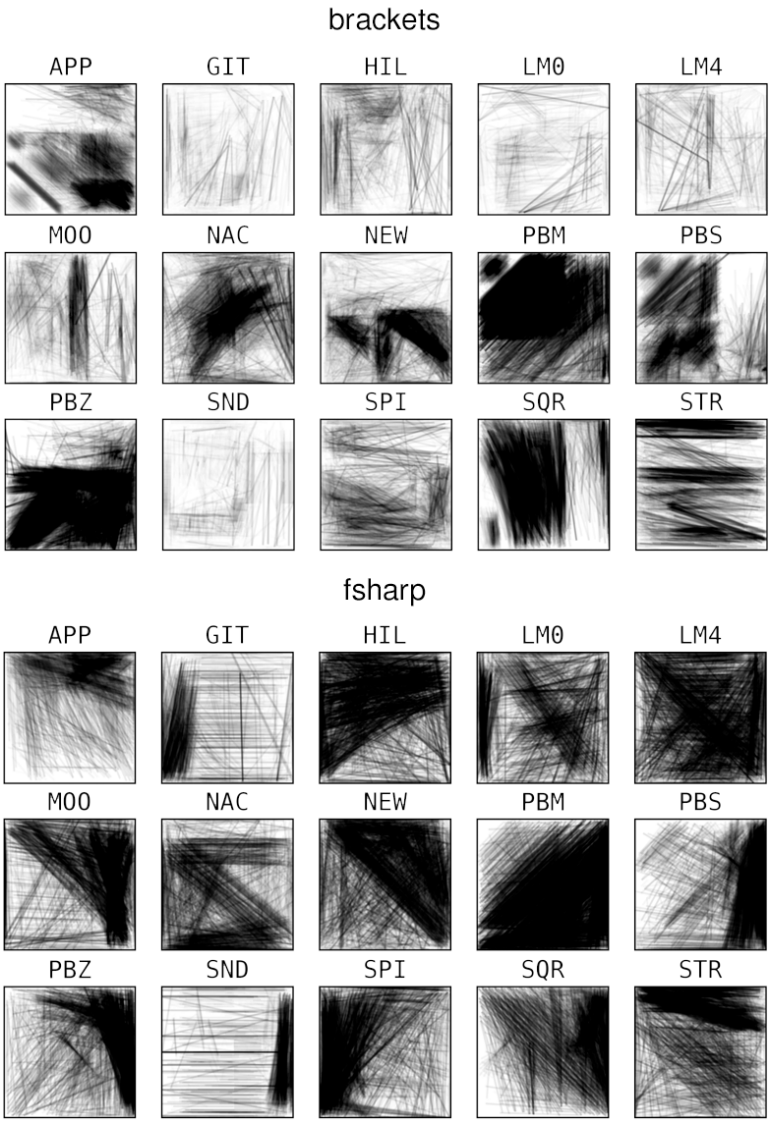


Figure 4.7: Instability (cell center motion) patterns, all methods, *brackets*, and *fsharp* datasets.

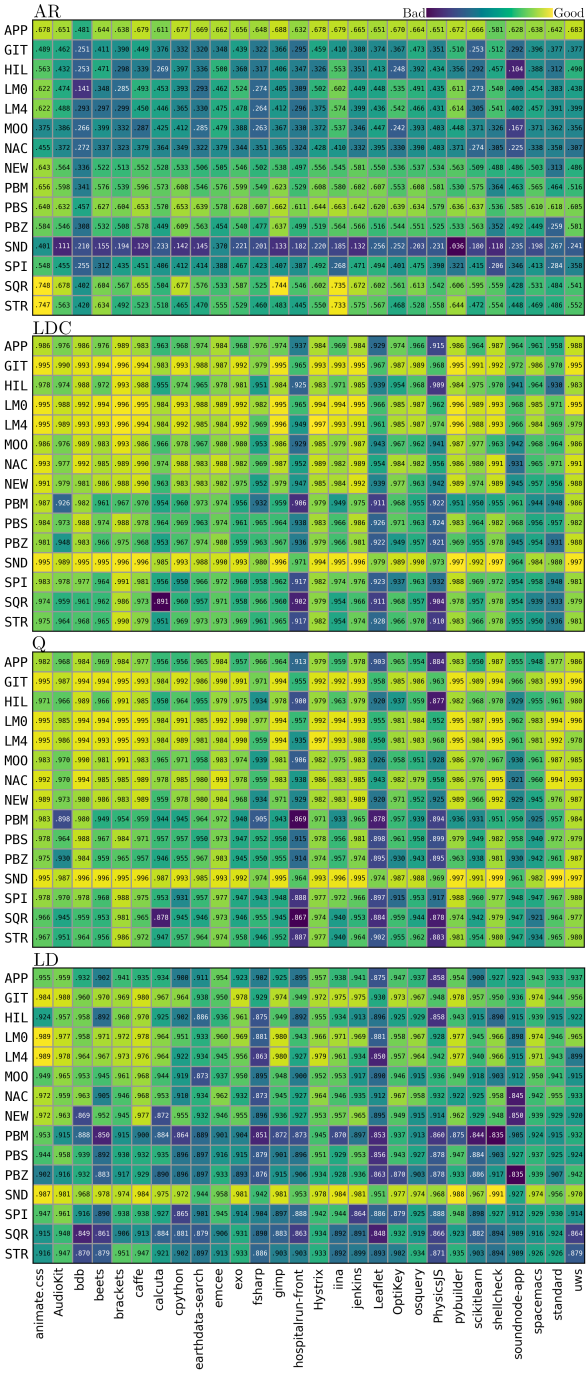


Figure 4.8: Average metric values for all techniques and all datasets.

stability. SND is the opposite outlier, scoring best on stability but clearly poorest on spatial quality. A middle group of methods (GIT, LM0, LM4, MOO, NAC, SPI, and HIL) trades well stability *vs* spatial quality. Within these, GIT scores the best stability, and LM0 the best spatial quality. As such, GIT and LM0 can be considered complementary methods with respect to the stability *vs* spatial quality trade-off. However, LM0 has a considerably more complex and slower implementation than GIT – for details, we refer to (Sondag et al., 2017). Separately, we see that GIT's star size (convex hull containing the lines emerging from the GIT point) is one of the smallest of all tested methods. Hence, GIT offers one of the most consistent behaviors over the entire dataset collection from all tested algorithms.

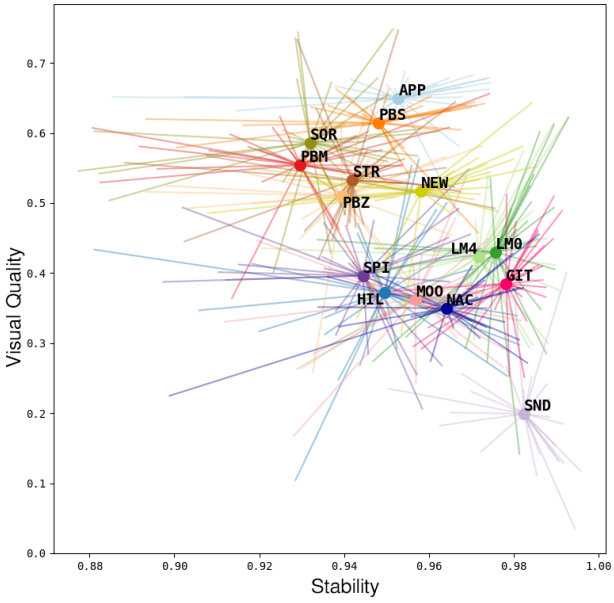


Figure 4.9: Star plot summarizing both visual quality and stability, all evaluated algorithms, all datasets.

4.6 CONCLUSION

We have presented Greedy Insertion Treemaps (GIT), a new method for computing treemap layouts for time-dependent hierarchies. As discussed earlier, there are only a few methods in the literature that consider quality aspects pertaining to *both* spatial quality and stability of such treemaps. Our contribution, in brief, is proposing a new method that takes both these quality aspects into account; and evaluating our method comprehensively on a broad dataset of 28 time-dependent hi-

erarchies extracted from real-world dynamic datasets (software repositories), against 14 well-known treemapping methods, and using 4 quality metrics. Our results show that our new GIT method strikes a good balance between spatial quality and stability as compared to state-of-the-art methods. Additionally, our method is simple to implement, fast, generic (with respect to the considered dynamic hierarchies), and has no hidden free parameters. More importantly, our method is an addition of a very small set of so-called *stateful* methods that consider the evolution of a dynamic tree sequence when computing suitable treemaps thereof. Most existing treemapping methods are not designed to consider tree state, which arguably makes them suboptimal for handling inherently stateful datasets like dynamic trees.

Chapter 3 has included GIT in the comparison with other treemapping algorithms and, more importantly, on a far more diverse collection of dynamic hierarchical datasets extracted from more application domains than evolving software systems. The respective evaluation shows that GIT performs well also for such more diverse datasets. In contrast to that broader (but shallower) evaluation, we focus in this chapter on a narrower evaluation that considers only datasets from software systems, but analyze GIT's performance in more depth. This provides, we believe, additional insights that show that, for the domain of evolving software hierarchies, GIT is the best solution in terms of trade-off of stability and spatial quality. Drawing this conclusion for the wider collection of datasets evaluated in Chapter 3 is, however, not yet possible. To do this, we would need to explore GIT (and the other tested algorithms there) in more detail, *e.g.*, by visually examining the dynamic patterns they produce, as shown here in Fig. 4.6. This also underlines the fact that using only a few quantitative metrics such as stability and visual (spatial) quality is necessary, but not sufficient, to fully capture the behavior of dynamic treemapping algorithms. Future work is needed to capture more complex motion patterns, such as those shown in Fig. 4.6, by suitable metrics, and next to gauge how desirable (or, on the other hand, confusing for the end user) are such patterns when present in a dynamic hierarchy visualization. Finally, understanding the trade-off between the (algorithmic) reasons behind spatial quality and stability, *i.e.* what to do to optimally satisfy both these requirements, is an open problem, to which we believe to have contributed to with our current work.

5 EVALUATING DYNAMIC PROJECTIONS

The first part of this thesis has explored how to use treemapping algorithms to visualize hierarchical and time-dependent data. We have presented evaluations of existing treemap methods, and also introduced a new concept of stability which is key to characterizing dynamic treemaps. We have also presented a new dynamic treemapping algorithm that scores better than existing ones on the joint requirements of stability, spatial quality, simplicity, and computational scalability. However, treemaps only address the visual exploration for weighted time-dependent hierarchical data.
The second part of this thesis focus on time-dependent multidimensional data, i.e., *data with a large number or attributes or dimensions whose values change over time. To make sense of such multidimensional datasets, dimensionality reduction methods (or projections) represent one of the most popular and effective approaches. Similarly to the situation regarding the literature of* ***dynamic*** *treemaps, there are few works that consider* ***dynamic*** *projections, that aim to visualize time-dependent multidimensional data. Following the approach presented in Chapters 2-3, we attack this problem by presenting here an extensive evaluation for dynamic projection algorithms, also introducing new suitable ways to define and measure the stability of such algorithms in the presence of data changes.*

Abstract: Dimensionality reduction methods are an essential tool for multidimensional data analysis, and many interesting processes can be studied as time-dependent multivariate datasets. There are, however, few studies and proposals that leverage on the concise power of expression of projections in the context of dynamic/temporal data. In this chapter, we aim at providing an approach to assess projection techniques for dynamic data and understand the relationship between visual quality and stability. Our approach relies on an experimental setup that consists of existing techniques designed for time-dependent data and new variations of static methods. To support the evaluation of these techniques, we provide a collection of datasets that has a wide variety of traits that encode dynamic patterns, as well as a set of spatial and temporal stability metrics that assess the quality of the layouts. We present an evaluation of 9 methods, 10 datasets, and 12 quality metrics, and elect the best-suited methods for projecting time-dependent multivariate data, exploring the design choices and characteristics of each method. Additional results can be found in the online benchmark repos-

This chapter is based on the paper "Quantitative Evaluation of Time-Dependent Multidimensional Projection Techniques" (Vernier et al., 2020a)

itory. We designed our evaluation pipeline and benchmark specifically to be a live resource, open to all researchers who can further add their favorite datasets and techniques at any point in the future.

5.1 INTRODUCTION

Dimensionality reduction (DR) methods, also called projections, are used in many applications in information visualization, machine learning, and statistics. Compared to other high-dimensional data visualization techniques, projections are especially effective for datasets with many observations (also called samples or points) and attributes (also called measurements, dimensions, or variables) (Liu et al., 2017). Many projection techniques exist, with wide varieties of computational efficiency, ease of use, ability to preserve and/or enhance different data patterns. Surveys have also focused on assessing quantitative and qualitative aspects of projection techniques (Nonato and Aupetit, 2019; Van Der Maaten et al., 2009; Espadoto et al., 2019), thereby helping practitioners in choosing a suitable one for a given context.

Most projection techniques have been designed and evaluated only for *static* data. Projecting dynamic (time-dependent) data is, however, equally important. Such data is found in most science and engineering areas, such as biology (Teo et al., 2017), medicine (Grillenzoni and Fornaciari, 2019), and finance (Krapl, 2019). The body of research in time series visualization is rich (Aigner et al., 2008), thereby underlining the importance of visualizing such data. Yet, there are only few examples of projecting time-dependent data (Hu et al., 2010; Mao et al., 2007; Ward and Guo, 2011; Bernard et al., 2012; Nguyen et al., 2017; Jäckle et al., 2016). Even fewer works focus on designing projection techniques specifically for dynamic data (Rauber et al., 2016; Fujiwara et al., 2019). In particular, it is not clear how to measure *and* trade-off two key aspects of such projections: *visual quality* and *stability*. While visual quality was studied well for static projections, stability, seen as the ability to create a set of projections that allows users to maintain a cohesive mental map through time, is recognized as essential for dynamic data visualization (Archambault et al., 2011; Brehmer et al., 2019), but has not been formally defined nor quantified for dynamic projections.

We work towards filling this gap in assessing projection techniques for dynamic data with the following main contributions:

- We propose novel variations of existing static projection *techniques* for the context of visualizing time-dependent data;
- We propose a set of *metrics* to quantify the stability of dynamic projections;

- We *benchmark* the visual quality and stability of dynamic projections on a dataset collection to get insights on which methods favor which of the measured quality aspects.

Our work can help researchers in targeting the identified challenges of current dynamic projection techniques, therefore potentially leading to improved ones. Separately, practitioners can use our findings into the process of determining which dynamic projection technique is best suited to their given user context. Finally, our creation of an open benchmark for assessing dynamic projections (containing datasets, techniques, metrics, visualizations, and associated workflows) should benefit both user types by providing a basis via which such techniques can be transparently compared.

The structure of this chapter is as follows. Section 5.2 outlines related work and evaluation techniques for projections for static and dynamic data. Section 5.3 details the proposed experiment we conducted to quantitatively assess the behavior of projection techniques for dynamic data, including techniques, datasets, and evaluated metrics. Section 5.4 presents the obtained results. Section 5.5 discusses the causes of the observed dynamic projection behavior. Finally, Section 5.6 concludes the chapter. For replication purposes, all our datasets, code, workflow, and results are openly available (The Authors, 2019).

5.2 RELATED WORK

5.2.1 *Preliminaries*

We first introduce some notation. Let $\mathbf{x} \in \mathbb{R}^n$ be an n-dimensional sample. A revision $\mathbf{R}^t = \{\mathbf{x}_i^t\}$, or timestep, of our data consists of a set of N samples $\mathbf{x}_i^t$, $1 \leq i \leq N$ measured at the same time moment t. A dynamic dataset $\mathbf{D}$ is a list of T revisions $\mathbf{D} = \left\{\mathbf{R}^t\right\}, 1 \leq t \leq T$. For simplicity of exposition and implementation, but without loss of generality, we consider next that the sample count N is constant over time. In this case, $\mathbf{D}$ can be represented as a set of T N-by-n matrices, one for each timestep.

A projection technique is a function $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$, where $q \ll n$. For visualization purposes, $q \in \{2, 3\}$. Since 2D projections are by far the most commonly used, we next only consider the case $q = 2$. We denote the projection of observation $\mathbf{x}$ by $P(\mathbf{x})$. For each timestep t, let $P(\mathbf{R}^t)$ be the 2D scatterplot of all points in $\mathbf{R}^t$. Finally, let $P(\mathbf{D})$ be the set of T scatterplots for all timesteps of dataset $\mathbf{D}$. These can be rendered as animations, small multiples, trail sets, or other visualization encodings.

Visualization of high dimensional data (Liu et al., 2017) is a well studied topic populated with many techniques such as parallel coordinate plots (Inselberg and Dimsdale, 1990), table lenses (Rao and Card, 1994), scatterplot matrices (Becker et al., 1996), and dimensionality reduction (DR) methods (Nonato and Aupetit, 2019; Van Der Maaten et al., 2009;

Espadoto et al., 2019). From all these we next focus only on DR techniques, both for static and dynamic data, and evaluation methods for both of these technique classes.

5.2.2 *Techniques for static dimensionality reduction*

The body of research that encompasses static DR is large and spans the fields of Information Visualization and Machine Learning. There are dozens of static techniques designed to optimize different objectives and to work well under different constraints. These can be classified and categorized using several taxonomies (Van Der Maaten et al., 2009) that guide users in choosing methods that meet their requirements. We do not further elaborate on such techniques, as several surveys extensively discuss static projections. Fodor (2002) present, to our knowledge, the first survey of DR techniques covering non-linear, vector quantization, and deep learning methods. Yin (2007) surveys non-linear DR methods. Bunte et al. (2012) proposes a framework to quantitatively compare nine DR methods. Cunningham and Ghahramani (2015) presents a theoretical comparison of 15 linear DR techniques. A similar survey, extended to 30 DR techniques, both linear and non-linear, is provided by Sorzano et al. (2014). Additional surveys look at DR methods in the larger context of high-dimensional data visualization, thus comparing and contrasting them with other visualization techniques (Buja et al., 1996; Hoffman and Grinstein, 2002; Engel et al., 2012; Kehrer and Hauser, 2013). The most recent survey in this area (Nonato and Aupetit, 2019) discusses technical aspects of DR methods, and also how such methods satisfy various user-level tasks.

5.2.3 *Evaluations of* static *dimensionality reduction*

Taxonomies as the ones listed above, compare DR methods mainly from technical (algorithmic) and task-suitability aspects. An increasingly visible alternative approach is to compare techniques by measuring various quality *metrics* on several techniques and datasets. A wealth of such quality metrics exist – for recent overviews, see (Pölzlbauer, 2004; Lee and Verleysen, 2009; Lueks et al., 2013; Nonato and Aupetit, 2019; Espadoto et al., 2019). Different metrics gauge different desirable aspects of a projection, and usually, several metrics are jointly used to assess DR quality (Gisbrecht and Hammer, 2015). Just as for DR techniques, metrics can be organized using different taxonomies. Following (Espadoto et al., 2019), these are as follows. *Aggregate* metrics, such as trustworthiness, continuity, neighborhood hit, distance and class consistency (Sips et al., 2009; Tatu et al., 2010), cluster visual separation metrics (Albuquerque et al., 2011; Sedlmair et al., 2013; Sedlmair and Aupetit, 2015), and metrics that capture human perception based on

machine learning (Aupetit and Sedlmair, 2016) characterize an entire 2D scatterplot by a single scalar value. This is convenient when comparing (many) different scatterplots to choose a suitable one, such as in scagnostics applications. However, a scatterplot may exhibit different quality values in different areas, so a single aggregated value may not be suitable (Joia et al., 2011; Nonato and Aupetit, 2019). *Point pair* metrics address this by measuring how point pairs $(P(\mathbf{x}), P(\mathbf{y}))$ in a projection relate to their corresponding sample pairs $(\mathbf{x}, \mathbf{y})$. These include Shepard diagrams (Joia et al., 2011) and co-ranking matrices (Lee and Verleysen, 2009). Finally, *local* metrics gauge separately every (small) neighborhood in a projection, thus providing the highest level of detail, and are typically visualized atop of the projection itself. These include the projection precision score (Schreck et al., 2010), stretching and compression (Aupetit, 2007; Lespinats and Aupetit, 2011), and false neighbors, missing neighbors, and average local errors (Martins et al., 2014, 2015).

Since all the above metrics aim to capture spatial aspects of the projection, we refer to them next as spatial quality metrics. Recent surveys have proposed extensive evaluations of spatial quality metrics on benchmarks containing a variety of datasets and DR methods (Espadoto et al., 2019; Van Der Maaten et al., 2009). However, time-dependent datasets were not considered.

5.2.4 *Techniques for* dynamic *dimensionality reduction*

The literature is much less rich regarding DR methods that *explicitly* consider dynamic data. The dynamic t-SNE (dt-SNE) method of Rauber et al. (2016) extends the well-known t-SNE method (van der Maaten and Hinton, 2008) by adding a stability factor λ to the objective function. Such a factor jointly minimizes the Kullback-Leibler divergence proposed by t-SNE to preserve high-dimensional point neighborhoods and also restricts the amount of motion $\|P(\mathbf{x}^{t+1}) - P(\mathbf{x}^t)\|$ that points can have between consecutive timesteps. More recently, Fujiwara et al. (2019) proposed a PCA-based method to deal with streaming data. Note that this is a harder (and different) problem from the one we aim to study since one cannot anticipate changes occurring upstream in the data when optimizing for placement of points in 2D. As such, analyzing this (and similar) methods is out of our scope. Separately, several authors use DR methods to create static maps that describe multivariate time series. Hu et al. (2010) use Self-Organizing Maps (Kohonen et al., 2001) to create 2D trails that capture the dynamics of human motion data. Rauber et al. (2017b) use similar trails, created by dt-SNE, to visualize the learning process of a neural network. Mao et al. (2007) use PCA to project text feature evolution in text sequences. Ward and Guo (2011), Bernard et al. (2012) and, more recently, Ali et al. (2019) use similar approaches to find cyclic behavior, outliers, and trends in temporal data

from medical, financial, and earth sciences domains. In contrast to the previous methods, m-TSNE (Nguyen et al., 2017) describes multivariate time series at a higher level of aggregation as single points instead of trails or polylines. Temporal MDS (Jäckle et al., 2016) projects $\mathbf{D}$ as a series of 1D projections, creating a map where the x-axis is time, and the y-axis shows the similarity of observations.

5.2.5 *Evaluation of dynamic dimensionality reduction*

Evaluating dynamic DR methods can be split into two aspects. First, just like for static DR methods, one is interested to see how well techniques capture the *spatial* aspects of the underlying data. For this, one typically uses the same types of spatial quality metrics as for static projections (Sec. 5.2.3). A separate important aspect for dynamic DR methods is *stability*. Loosely put, stability describes how a dynamic DR technique encodes *changes* in the data into *changes* in the 2D metaphor used to visualize the data – note that this definition of stability is the same as the one we used in the first part of this thesis for dynamic treemapping techniques. Such 2D metaphors can be grouped into spatial ones, where different timesteps map to different plots, such as in small multiples; and animation-based ones, where different timesteps are encoded into frames of a 2D animation.

Stability metrics were proposed and evaluated to assess the quality of other visualizations of dynamic data such as time-dependent treemaps (Sondag et al., 2017; Vernier et al., 2018a,b) (see also Chapters 2-3). Stability is related to the capacity of a DR technique to deal with so-called out-of-core data. Simply put, this means the ability for a projection, created from a given dataset $\mathbf{D}$, to add extra points $\mathbf{X} \notin \mathbf{D}$ to the resulting 2D depiction $P(\mathbf{D})$, without distorting this depiction too much so that its understanding becomes hard. While recent works consider out-of-core and stability as key properties for DR projections (Nonato and Aupetit, 2019; Boytsov et al., 2017; Espadoto et al., 2020b; García-fernández et al., 2013; Buja et al., 2008), we are not aware of specific quality metrics that quantify these.

5.3 EXPERIMENTAL SETUP

To evaluate how dynamic DR techniques perform, we follow a methodology similar to the one proposed in Espadoto et al. (2019) for evaluating static DR techniques, as follows. We first select a set of dynamic DR *techniques* to evaluate. Next, we select a collection of *datasets* that cover various aspects, or *traits*, that characterize high-dimensional dynamic data. Thirdly, we evaluate both spatial quality and stability *metrics* on all combinations of techniques and datasets; in this step, we also propose novel metrics to gauge stability. We describe all these steps next.

The analysis of the discovered correlations between techniques, dataset traits, and quality metrics obtained from our experiments is discussed afterwards in Sec. 5.4.

5.3.1 *Techniques*

We selected the dynamic DR techniques to evaluate based on the following considerations. First, we only consider techniques P, which, given a dataset consisting of several timeframes $\mathbf{R}^t$, produce corresponding 2D scatterplots $P(\mathbf{R}^t)$. We argue that this is the most generic definition of a dynamic projection – from such scatterplots, other types of visualizations can be constructed next as desired (animation, small multiples, trails). This is analogous to expecting a generic static projection technique to deliver a 2D scatterplot. Hence, techniques that deliver different output types, such as m-TSNE (Nguyen et al., 2017) and temporal MDS (Jäckle et al., 2016), are excluded from our evaluation. Secondly, we only consider techniques that (1) are generic with respect to the input data (size, dimensionality, provenance) they can handle; (2) well-known and often used in practice, so their evaluation arguably serves a sizeable user group; and (3) easy to set up, control, and have publicly available implementations, for reproducibility. We next describe the selected techniques.

t-SNE and variants: Probably the simplest way to project dynamic data is to compute a single, global, projection $P(\mathbf{D})$ for the entire dataset $\mathbf{D}$ and next visualize the timeframes by using the desired method, be it animation, trails, or small multiples. We next call this the *global* (G) approach. While this arguably favors stability (since P sees all data $\mathbf{D}$ at once), it likely yields limited spatial quality, since P has the challenging task of placing well *all* points from *all* revisions in $\mathbf{D}$. An equally simple approach is to compute independent projections $P(\mathbf{R}^t)$ for each revision $\mathbf{R}^t$. We call this next the *per-timeframe* (TF) approach. This arguably favors spatial quality, since P must only optimize positions for each revision $\mathbf{R}^t$ separately, rather than the entire $\mathbf{D}$. However, this approach can yield poor stability, since timeframes are projected without knowledge of each other. Both the global and timeframe approaches were suggested, but not quantitatively evaluated, in the dt-SNE paper (Rauber et al., 2016). Given this, and also the fact that t-SNE is a very well-known static technique, we next consider G-t-SNE, TF-t-SNE, and dt-SNE in our evaluation.

UMAP: This recent DR technique (McInnes et al., 2018) has a mathematical foundation on Riemannian geometry and algebraic topology. According to recent studies (Espadoto et al., 2019; Becht et al., 2019), UMAP offers high-quality projections with lower computational cost and better global structure preservation than t-SNE, being thus an

interesting competitor in the DR arena. We consider in our evaluation both the global (G-UMAP) and per-timeframe (TF-UMAP) variants of this technique.

PCA: Following (Fujiwara et al., 2019; Mao et al., 2007; Ward and Guo, 2011), we also consider Principal Component Analysis (Jolliffe, 1986), implementing the global and timeframe strategies. In detail, PCA performs a linear mapping of the data $\mathbf{D}$ to, in our case, 2D by maximizing the data variance in the 2D representation. The global strategy implies computing PCA once for the entire $\mathbf{D}$. In contrast, timeframe PCA means computing PCA separately for each revision $\mathbf{R}^t$. Given the widespread use of PCA in many fields of science, and also its out-of-core ability (which, as outlined in Sec. 5.2.5, is related to stability), we consider both G-PCA and TF-PCA next in our evaluation.

Autoencoders: Often used in dimensionality reduction and representation learning, autoencoders (Hinton and Salakhutdinov, 2006; Ballard, 1987) are hourglass-shaped neural networks. They are composed of an encoder that takes the original data $\mathbf{D}$ and compresses it into a compact (latent) representation $P(\mathbf{D})$ of lower dimensionality (two in our case), and a decoder, which takes $P(\mathbf{D})$ and aims to reconstruct a good approximation of the original data $\mathbf{D}$. While autoencoders have been often used to create static projections of high-dimensional data, they have not, to our knowledge, been quantitatively evaluated for their ability to handle dynamic data. We evaluated four types of autoencoders, as follows. *Dense autoencoders* (AE) are comprised of only fully-connected (dense) layers and are the standard variant. *Convolutional autoencoders* (CAE) (Masci et al., 2011) have both fully-connected and convolutional layers. The convolutional layers apply a non-linear transformation to the data that takes into account the spatial correlation between attributes, for instance, the proximity of pixels in an image. *Variational autoencoders* may have both fully-connected layers (VAE) (Kingma and Welling, 2014) and convolutional layers (CVAE). The main difference between dense and variational autoencoders is the addition of stochastic behavior in the intermediate layer of the latter. The encoder produces two vectors – an intermediate representation (IR) and an uncertainty degree σ for each IR value. The decoder tries to reconstruct the input through a sample from the latent space distribution with mean IR and variance σ, thus forcing the network to learn similar representation for similar inputs. Convolutional based architectures are not generic regarding input and a meaningful spatial relationship between attributes is expected (such as found on image data). We, therefore, restrain the analysis on this document to AE and VAE. The results of CAE and CVAE runs for the image based datasets (fashion and quickdraw) can be found on the online material (The Authors,

Table 3: Hyperparameters of the autoencoder-based DR methods

dataset	technique	# hidden layers	# nodes/layer	# epochs
cartolastd	AE	2	10, 10	50
cartolastd	VAE	2	10, 10	100
cifar10cnn	AE	2	10, 10	20
cifar10cnn	VAE	2	100, 10	20
esc50	AE	2	10, 10	40
esc50	VAE	2	100, 10	20
fashion	AE	3	500, 500, 2000	40
fashion	VAE	3	2048, 1024, 512	20
gaussians	AE	2	10, 10	20
gaussians	VAE	2	100, 10	20
nnset	AE	2	10, 10	20
nnset	VAE	2	100, 10	20
qtables	AE	2	10, 10	20
qtables	VAE	2	100, 10	20
quickdraw	AE	3	500, 500, 2000	40
quickdraw	VAE	3	2048, 1024, 512	20
sorts	AE	2	10, 10	20
sorts	VAE	2	100, 10	20
walk	AE	2	10, 10	20
walk	VAE	2	100, 10	20

2019).

Implementation: We implemented the chosen dynamic DR techniques (G-t-SNE, TF-t-SNE, dt-SNE, G-UMAP, TF-UMAP, G-PCA, TF-PCA, AE, CAE, VAE, CVAE) as follows. For t-SNE and PCA, we used scikit-learn (Pedregosa et al., 2011) with default parameters. For dt-SNE and UMAP, we used the implementation provided online by the authors (Rauber et al., 2016; McInnes et al., 2018). Finally, we implemented the four autoencoder models using Keras (Chollet et al., 2015), with different numbers of layers, nodes per layer, optimizers, and training routines. Tab. 3 shows the values, for each autoencoder and dataset, that delivered the best results, and which we used next. The code, notebooks, and instructions to recreate our results are available online (The Authors, 2019).

5.3.2 *Datasets*

There is, to our knowledge, no standardized benchmark for evaluating DR techniques. Espadoto et al. (2019) took a first step towards providing such a benchmark containing 19 datasets. However, all these are time-independent, thus not suitable for us. We followed here a similar approach, *i.e.* collecting a set of 10 high-dimensional and dynamic datasets that exhibit significant variations in terms of provenance, num-

ber of samples N, number of timesteps T, dimensionality n, intrinsic dimensionality ρ_n (percentage of n dimensions that explain 95% of the data variance), and sparsity ratio σ_n (percentage of zeros in the data). All datasets are labeled into 3 to 10 classes. We only use labels for visualization and quality assessment and not the projection itself. Table 4 shows the characteristics, or traits, for these datasets. Further details on them are listed below.

- **cartolastd:** Player statistics for the second turn of the 2017 Brazilian football championship. Data was extracted from an open-source project (Gomide and Gualberto, 2019) that scrapes the Cartola FC football platform. Each timestep corresponds to a tournament round. Variables relate to per-match performance of a given player (number of goals, assistances, fouls, defenses, etc.). Players are labeled by their playing position (goalkeeper, right or left-back, defender, midfielder, forward).

- **cifar10cnn:** Last hidden layer activations after each training epoch for a convolutional network trained to classify the CIFAR10 (Krizhevsky, 2009) dataset.

- **esc50:** Sound samples of 8 classes (brushing teeth, chainsaw, crying baby, engine, laughing, rain, siren, wind) compressed to 128 frequencies and smoothed over time. Extracted from Piczak's ESC50 dataset (Piczak, 2015).

- **fashion:** 100 images from each of the 10 classes (T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot) of the FashionMNIST (Xiao et al., 2017) dataset with decreasing amounts of noise over time.

- **gaussians:** Synthetic dataset used to evaluate dt-SNE (Rauber et al., 2016). Isotropic gaussian blobs in nD with diminishing spread over time.

- **nnset:** Internal states (weights and biases) of several neural networks during training with the MNIST dataset (LeCun and Cortes, 2010). The networks have the same architecture but use different optimizers, batch sizes, and training-set sizes.

- **qtables:** Internal state of agents learning to move a car up a hill using the reinforcement learning algorithm Q-learning. The classes represent variations of learning rates and discounts.

- **quickdraw:** Drawing sequences for 600 objects of 6 different classes drawn by random people. Extracted from the "Quick, Draw!" Google AI experiment (Jongejan et al., 2016).

Table 4: Datasets and their traits used in the evaluation.

dataset	samples N	timesteps T	dimensions n	classes	intrinsic dim. ρ_n	sparsity ratio σ_n
cartolastd	696	19	17	5	0.6470	0.0000
cifar10cnn	1000	30	10	10	0.6599	0.0000
esc50	320	108	128	8	0.0345	0.0000
fashion	1000	10	784	10	0.4762	0.2971
gaussians	2000	10	100	10	0.3680	0.0000
nnset	80	30	8070	8	0.0057	0.0001
qtables	180	40	1200	9	0.0077	0.0007
quickdraw	600	89	784	6	0.4309	0.9013
sorts	80	100	100	8	0.3505	0.0100
walk	300	50	100	3	0.4783	0.0001

- **sorts:** This dataset was designed to compare the behavior of eight sorting algorithms. The algorithms sort different arrays of 100 random values in $[0, 1]$. As they do so, we take snapshots of the intermediate states, until sorting is over. Each observed point is an (algorithm, array) run, and its feature vector is the partially sorted array at a given time.

- **walk:** Synthetic dataset with similar structure to *gaussians*. It contains 3 high-dimensional clusters oscillate (approach, intermingle and cross, and then drift apart) in $\mathbf{R}^{100}$ over time. We designed this dataset to see how well the studied DR techniques can capture the approaching, mingling, and drifting-away dynamics mentioned above.

Covering all variations of high-dimensional datasets with a benchmark is already daunting for static data (Espadoto et al., 2019), thus even more for dynamic data, as there are many types of dynamic patterns possible. Hence, we cannot claim that our benchmark is *exhaustive* in terms of the space it samples. However, we believe that the included datasets exhibit a rich variety of different traits (Tab. 4). Also, no two datasets are redundant, *i.e.*, have all traits similar. Given that, to date, no other benchmark exists for this task, we believe ours is a good start in supporting the intended evaluation.

5.3.3 *Metrics*

We measure the quality of all projection techniques (Sec. 5.3.1) on all datasets (Sec. 5.3.2) using both spatial quality and stability metrics, similarly to other evaluations of multivariate dynamic data visualizations such as treemaps (Sondag et al., 2017; Vernier et al., 2018a,b). In our evaluation, we use the same metrics as the survey (Espadoto et al., 2019) (and a few extra ones) over all revisions $\mathbf{R}^t$, as follows.

5.3.3.1 *Spatial metrics*

Neighborhood preservation (S_{NP}): With values in $[0, 1]$, with 1 being the best, this is the percentage of the k-nearest neighbors of $\mathbf{x} \in \mathbf{D}$ that project in the k-nearest neighborhood of $P(\mathbf{x})$.

Neighborhood hit (S_{NH}): With values in $[0, 1]$, with 1 being the best, this is the fraction of the k-nearest neighbors of a projected point $P(\mathbf{x})$ that have the same class label as $P(\mathbf{x})$. Since we know that our datasets exhibit reasonably well-separated classes in $\mathbb{R}^n$, a proper DR technique (from the perspective of class separation tasks) should yield a high neighborhood hit.

Trustworthiness (S_{Trust}): With values in $[0, 1]$, with 1 being the best, this measures how well the k nearest neighbors $NN^k(P(\mathbf{x}))$ of a projected point $P(\mathbf{x})$ match the k nearest neighbors $NN^k(\mathbf{x})$ of a data point $\mathbf{x}$. Simply put, trustworthiness measures how few missing neighbors (Martins et al., 2014) a projected point has. Formally, if $U^k(\mathbf{x})$ is the set of points that project in $NN^k(P(\mathbf{x}))$ but are not in $NN^k(\mathbf{x})$, and $r(\mathbf{x}, \mathbf{y})$ is the rank of $\mathbf{y}$ in the ordered set of nearest neighbors $NN^k(P(\mathbf{x}))$, trustworthiness is then defined as $1 - \frac{2}{Nk(2N-3k-1)} \sum_{x=1}^{N} \sum_{y \in U^k(\mathbf{x})} (r(x, y) - k)$.

Continuity (S_{Cont}): With values in $[0, 1]$, with 1 being the best, this measures how many missing neighbors (Martins et al., 2014) a projected point has. Following the above notations, let $V^k(\mathbf{x})$ be the points that are in $NN^k(\mathbf{x})$ but do not project in $NN^k(P(\mathbf{x}))$. Let also $\hat{r}(\mathbf{x}, \mathbf{y})$ be the rank of $\mathbf{y}$ in the ordered set of neighbors $NN^k(\mathbf{x})$. Continuity is then defined as $1 - \frac{2}{Nk(2N-3k-1)} \sum_{x=1}^{N} \sum_{y \in V^k(\mathbf{x})} (\hat{r}(x, y) - k)$.

In contrast to Espadoto et al. (2019), we compute neighborhood preservation, trustworthiness, and continuity for multiple (20) neighborhood sizes equally spread between $k = 1\%$ and $k = 20\%$ of the point count N. Similarly, for the neighborhood hit, we use 20 values for k, ranging from 0.25% to 5%. This allows us next to study the spatial quality of projections at different scales (Martins et al., 2015).

Normalized stress (S_{Stress}): With values in $\mathbb{R}^+$, lower meaning better distance preservation, stress measures the pairwise difference of distances of points in nD and qD. We define S_{Stress} as $\sum_{ij} \left(d_{ij}^t - \overline{d_{ij}^t} \right)^2 / \sum_{ij} (d_{ij}^t)^2$, where d_{ij}^t and $\overline{d_{ij}^t}$ are the Euclidean distances between data points $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$, and between their projections $P(\mathbf{x}_i^t)$ and $P(\mathbf{x}_j^t)$, respectively, for $1 \leq t \leq T$, for every point pair (i, j). To ease analysis, we scale distances using standardization.

Shepard diagram metrics: The Shepard diagram is a scatterplot of d_{ij} by $\overline{d_{ij}}$, for every pair (i, j) in $\mathbf{D}$ (see Fig. 5.4b). It visually tells how different ranges of distances between points are affected by a projection. Plots close to a diagonal indicates good distance preservation. Deviations from this highlight patterns such as poor preservation of long/short distances, creation of false neighborhoods, or stretching and compression of the manifold on which the data is defined (Joia et al., 2011). We summarize and quantify Shepard diagrams by measuring the relationship between the two distances. Following Espadoto et al. (2019), we use Pearson correlation to measure the linearity of the relationship, and we add Spearman and Kendall correlation to measure the monotonicity of the relationship. The three resulting correlation metrics $S_{Pearson}, S_{Spearman}, S_{Kendall}$ range from -1 to 1, where 1 means perfect positive correlation.

5.3.3.2 *Temporal stability metrics*

As previously stated, there are no metrics in the literature specially designed to measure the temporal stability of DR methods. We next propose two such metrics, as follows. The two variables whose relationship we want to measure are the *change of the attributes* of a sample $\mathbf{x}$ from time t to $t+1$, measured as the nD Euclidean distance $\delta^t = \|\mathbf{x}^t - \mathbf{x}^{t+1}\|$, and *movement of the projection point* $P(\mathbf{x})$ from time t to $t+1$, measured as the 2D Euclidean distance $\overline{\delta^t} = \|P(\mathbf{x}^t) - P(\mathbf{x}^{t+1})\|$. Ideally, for a temporally stable P, we want $\overline{\delta^t}$ to be proportional to δ^t. However, this may be a too hard constraint for P to satisfy, just as perfect nD to 2D distance preservation is hard to achieve for static projections. A more relaxed requirement for a temporally stable P is to have $\overline{\delta^t}$ a monotonic increasing function of δ^t. Indeed, if this constraint were not obeyed by P, then if an observation $\mathbf{x}^t$ changes only slightly over time, its projection $P(\mathbf{x}^t)$ could move a lot. That is, if $\delta^t \ll \overline{\delta^t}$, the projection P is unstable, and would convey the user the wrong impression that data is changing a lot. Conversely, if $\mathbf{x}^t$ strongly changes over time, but $P(\mathbf{x}^t)$ remains roughly static, *i.e.* if $\delta_i^t \gg \overline{\delta_i^t}$, then the user gets the wrong impression that the data is not changing. Hence, for a temporally stable P, the two changes $\overline{\delta^t}$ and δ^t should be positively correlated.

To measure the relationship of δ^t and $\overline{\delta^t}$, we adapt the static spatial quality metrics introduced in Sec. 5.3.3.1 as follows:

Normalized temporal stress (T_{Stress}): We define temporal stress as $\sum_{i\,t} (\delta_i^t - \overline{\delta}_i^t)^2 / (\delta_i^t)^2$, where the subscript i indicates sample point $\mathbf{x}_i$. As for the spatial normalized stress, we normalize distances using standardization. Low stress values indicate that the 2D changes $\overline{\delta^t}$ reflect closely their nD counterparts δ^t, which is desirable.

Temporal Shepard diagram metrics: Akin to the spatial metrics defined on Shepard diagrams, we measure the Pearson, Spearman, and Kendall correlations $T_{Pearson}$, $T_{Spearman}$, $T_{Kendall}$ between δ and $\overline{\delta}$ for every observation and consecutive timesteps. High correlation values indicate that the 2D changes $\overline{\delta^t}$ are strongly correlated with their nD counterparts δ^t, which is desirable.

At this point, it is further interesting to follow the parallel of the concept of stability as defined for dynamic treemaps (Chapter 3) and as defined here above for dynamic projections, respectively. As indicated in Sec. 5.2.5, at a conceptual level, the two forms of stability are identical – they both measure how much change in the visualization (treemap or 2D projection, respectively) follows the change in the data (hierarchy or nD dataset, respectively). However, there are some important technical differences. For dynamic hierarchies, data change in a hierarchy is hard to measure (see the discussion in Sec. 3.3.2), and it resides in a different space than the visual (treemap cell) change, making it hard to relate the two in a single formula. As such, for defining treemap stability, we use a proxy method based on the so-called baseline treemap. For projections, the situation is different: Both nD (data) and 2D (projection) spaces are of the same nature, meaning, we can quantify change in both by using Euclidean distances. Hence, we can relate both changes in a single formula (or formulas) as done above with the normalized temporal stress and temporal Shepard diagram metrics.

5.4 EVALUATION AND RESULTS

We evaluate the 12 quality metrics introduced in Sec. 5.3.3 on all (dataset, method) pairs formed by the selected 9 DR methods and 10 datasets, and analyze next the results. We do this by proposing several metric visualizations, from highly aggregated (to help forming first insights) to detailed (to examine more subtle points). For a direct impression, see also the videos showing the actual dynamic projections in action, available online at (The Authors, 2019).

5.4.1 *Aggregated results*

Figure 5.1 shows average metric values computed over all datasets and techniques. Light colors represent high metric values (preferred). The colormap in Fig. 5.1 was normalized independently by the min and max of each column (metric), and it was inverted for the stress-based metrics, as low values mean preferred results for these. At the bottom of each cell, a 1D scatterplot with density mapped to luminance shows the distribution of the values of the (metric, method) pair corresponding to that cell over all datasets. The red line shows the distribution mean. The table in Fig. 5.1 is divided into three blocks: The two left blocks show spatial

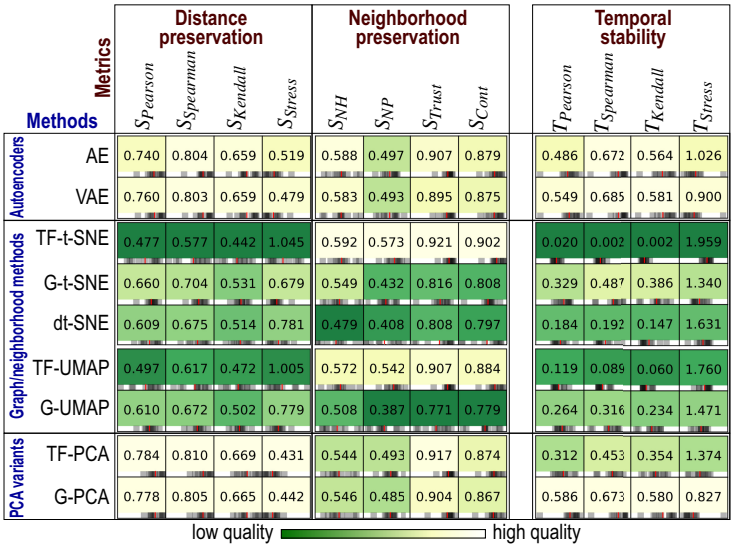


Figure 5.1: Aggregated metric results over all datasets.

metrics for distance and neighborhood preservation, respectively. The right block shows stability metrics.

Figure 5.1 helps us to find methods that strike a balance between spatial quality and stability. In this sense, (variational) autoencoders and G-PCA score, overall, the best. The other methods are good in one aspect but not the other: Timeframe t-SNE has high neighborhood metric values but poor distance preservation and the poorest stability from all assessed methods. Timeframe PCA has high distance preservation but relatively low stability. dt-SNE appears to be as good spatially as G-t-SNE, but slightly less stable. This is an interesting finding since dt-SNE was explicitly designed (but not quantitatively assessed) to aid stability.

5.4.2 *Dataset-wise results*

Figure 5.1 is simple to read but heavily aggregated, so it does not show how the quality of specific methods depends on specific *datasets*. To see this, Fig. 5.2 shows all metric results for all datasets without aggregation. As in Fig. 5.1, light colors mean good results. Columns are now not normalized. Column groups (a-f) represent spatial metrics, and columns (g-h) represent stability metrics. We use different quantitative colormaps to indicate different types of measured data. By examining Fig. 5.2, we obtain the following insights:

Unstable methods: TF-t-SNE is always unstable regardless of the dataset. This refines the instability finding over TF-t-SNE (Sec. 5.4.1) by

showing that this occurs irrespective of the dataset. Also, it confirms the same observation in Rauber et al. (2016), which, however, was not quantitatively confirmed there. The reason for this instability is the stochastic nature of t-SNE, which strongly manifests itself if we run the method from scratch on every new revision (timeframe). We could attribute the instability of TF-UMAP to the same reason.

Poor spatial quality: G-t-SNE and G-UMAP score poorly on distance and neighborhood preservation on most datasets. This is the aforementioned difficulty (Sec. 5.3.1) of constructing a *single* projection covering many samples in many timeframes. This is much harder than constructing a projection that preserves only neighborhoods formed by points in a *single* timeframe. We see here again the trade-off between spatial quality and stability.

Neighborhood preservation: Here we see dataset-specific behavior: For *gaussians*, S_{NP}, S_{Trust}, and S_{Cont} peak at a neighborhood size of roughly 10% of the dataset size. This makes sense since this is the size of the clusters present in this dataset – when k exceeds this value, the metrics will start considering points in other clusters, thus decrease. More interestingly, we see some outliers (dark bands in the heat-colormapped plots). These are techniques that score poorly for any k value. Among these, we find G-t-SNE, dt-SNE, and G-UMAP. At the other extreme, TF-t-SNE and TF-UMAP score the best results at neighborhood preservation, followed by AE, VAE, G-PCA, and TF-PCA.

Dynamic t-SNE: In contrast to the good results qualitatively observed on the single *gaussians* dataset showed in Rauber et al. (2016), dt-SNE performs less well in both spatial quality and stability for several other of the considered datasets, being quality-wise somewhere between TF-t-SNE and G-t-SNE for all considered metrics.

Dataset difficulty: Some datasets are considerably harder to project with good quality than others, no matter which technique we use. For example, *walk* has poor stability for all techniques. In contrast, *gaussians* has good stability for all techniques (except the t-SNE and UMAP variants) and good neighborhood preservation for all techniques. To study how dataset characteristics influence quality, we compute the correlation of the distance-preservation, neighborhood, and temporal stability metrics (measured over all techniques) with the six traits that we used to characterize our datasets (Tab. 4). Table 5 shows the results. A few things stand out: As the number of samples N increases, the difficulty to preserve distances also increases, but neighborhoods are preserved better. Conversely, as sparsity σ_n increases, it becomes harder to preserve neighborhoods. Separately, we do not find any strong (positive or negative) correlation of temporal stability with any of the traits.

Overall, this suggests that the traits are useful in predicting *spatial* quality of projections. However, we need additional traits that capture the data dynamics to reason about the projections' temporal stability.

Table 5: Correlation between metric types and dataset traits.

	samples N	timesteps T	dimensions n	classes	intrinsic dim. ρ_n	sparsity ratio σ_n
distance preservation	-0.429566	0.145921	-0.076177	-0.285476	-0.007806	-0.211705
neighborhood preservation	0.385248	-0.380503	-0.298868	0.243835	0.172121	-0.404517
temporal stability	0.150231	0.012017	-0.009754	0.275271	-0.085292	0.160295

5.4.3 *Fine-grained analysis*

While Fig. 5.2 shows all computed metrics for each (dataset, method) combination, metric values are still aggregated to a single scalar per combination. This does not show how metrics vary over the *extent* of a projection and/or over *time*. There are more patterns in dynamic projections than we can capture by a set of metrics, no matter how good these are. To get such insights, we next present a fine-grained analysis that aggregates the metrics even less (see Figure 5.4) for a single dataset (*cartolastd*, chosen as it is alphabetically the first in our benchmark). Similar visualizations for all other datasets in the benchmark are available online (The Authors, 2019). We next analyze these methods for this dataset from several perspectives, as follows.

Stability visual assessment: Figure 5.4a shows the actual dynamic projections with point trails $(P(\mathbf{x}_i^1), \ldots, P(\mathbf{x}_i^T))$, one per player i. Colors map the players' labels. This visualization already says a lot about the behavior and similarities of the studied DR methods (see also the submitted videos). The instability of TF-t-SNE and TF-UMAP becomes apparent, as their trails cover a very large area in the projection space. However, these methods achieve a quite good separation of same-label clusters. In contrast, dt-SNE shows trails that depict much local movement. Both PCA variants show relatively little movement, with points oscillating along two main axes, which are the main eigenvectors computed by the methods. At the other extreme, AE, VAE, and G-t-SNE show the least motion. However, this does not imply by itself a high quality: G-t-SNE, for instance, achieves indeed a better visual spreading of samples in the available projection space, but it has very poor neighborhood preservation (see G-t-SNE results in Fig. 5.2) and, as already discussed above, it also has very poor stability.

Distance preservation: Figure 5.4b shows the Shepard diagram of distances, which is a scatterplot of d_{ij} by $\overline{d_{ij}}$, for every pair (i, j) in $\mathbf{D}$, that helps us understand the distance preservation aspect of

each technique. We see that the AE and PCA variants have overall better distance preservation (plots closer to the diagonal) than the t-SNE/UMAP variants. Also, we see that AE and PCA typically *compress* nD distances to 2D (points mainly under the main diagonal), whereas the t-SNE/UMAP variants both compress and stretch these (points are located both under and above this diagonal).

Inspired by the Spearman and Kendall correlations, we consider next the agreement of *ranks* instead of aggregating it to a single value. Figure 5.4c shows this, for distance preservation, by a histogram of the *absolute* rank differences of nD and 2D distances between point pairs. In a projection with $S_{Spearman} = S_{Kendall} = 1$, such differences would be minimized, *i.e.*, the k^{th} largest 2D distance $\overline{d_{ij}}$ should correspond to the k^{th} largest nD distance d_{ij} for every point pair (i, j). In this case, all rank differences are zero, which would yield a histogram showing a single high bar at zero (left of the histogram). Significant rank differences spread the histogram to the right, showing poor monotonicity between the two variable ranks. From these plots, we see, again, that AE and VAE score the best, followed by G-PCA, TF-PCA, and then the t-SNE and UMAP variants.

Stability metrics: Figure 5.4d shows Shepard diagrams for the point movements, *i.e.*, scatterplots of δ by $\overline{\delta}$ for every sample compared to itself in the next timestep, for all timesteps. Note that, in these scatterplots, every point is a *sample*, whereas in the classical Shepard diagrams (Fig. 5.4b), every point is a *pair* of samples. Ideally, we want δ to be positively correlated to $\overline{\delta}$, which means a plot close to the main diagonal. The AE and PCA variants show the closest plots to the main diagonal, thus, best stability. At the other extreme, TF-t-SNE shows widely varying 2D change for similar nD change, thus, high instability. Finally, Figure 5.4e shows the absolute rank difference histograms for change. Their interpretation follows the one for the distance-preservation histograms (Fig. 5.4c): Left peaked histograms indicate high stability, whereas flatter ones indicate a discrepancy in 2D *vs* nD changes. These histograms strengthen the insights obtained so far, making it even clearer that the AE and G-PCA methods are far stabler than the t-SNE, UMAP and TF-PCA.

5.5 UNDERSTANDING DYNAMIC PROJECTION BEHAVIOR

The coarse-grained and fine-grained analyses presented so far highlighted that there are significant differences in the behavior of dynamic DR methods that depend on both the method and the dataset. In this process, we also saw that visual quality and stability seem to be, in general, mutually competing for concerns – methods that are good in one are not the best in the other. We further explore these observations as

follows. First, we analyze the causes of the observed (lack of) stability and link these to the way the studied DR techniques operate (Sec. 5.5.1). Next, we summarize all our findings and propose a workflow to assist the practitioner in selecting a suitable DR technique for projecting dynamic data (Sec. 5.5.2).

5.5.1 *Analysis of (un)stable behavior*

Beside empirically measuring and observing that different DR techniques have widely different stabilities, it is useful to analyze the *causes* of these differences, which we do next.

t-SNE and UMAP: Our results tell that TF-t-SNE and TF-UMAP, that is, projections computed independently for each timestep, are the most unstable of the assessed techniques. This is so since these are stochastic methods that optimize non-convex objective functions using randomly seeded gradient descent. Hence, different runs with the same data can create projections where different clusters might be formed and/or placed at different 2D positions. Figure 5.5a,b shows the last scenario. From timesteps 1 to 2 of the TF-t-SNE run of the *fashion* dataset, even though the local structure remains the same, the absolute position of the points and clusters changes drastically. In conclusion, using t-SNE/UMAP independently per timeframe is definitely not a good option for dynamic data.

dt-SNE: We encountered several cases where dt-SNE seems to have trouble optimizing its objective function – for details, see the videos for *qtables* and *sorts*. In both these cases, dt-SNE did not capture any of the spatial structures present in the data, nor produced any sensible movement. These visual findings can be confirmed by the dark lines (low-quality values) in Fig. 5.2. We also noticed that dt-SNE is very sensitive to the choice of hyperparameters. Concluding, whereas the initial findings in Rauber et al. (2016), obtained on a single dataset (*gaussians*) position this technique as a good option for projecting dynamic data, our additional findings raise questions about the practical value of this technique.

PCA: We also see instability in TF-PCA, but for different reasons than the ones discussed above. Specifically, if there is a change in rank of the top two eigenvectors from timestep t to the next one, *i.e.*, one of the associated eigenvalues becomes larger than the other, the projection exhibits an artifact that resembles a *reflection* – see the *quickdraw* dataset in the two timesteps in Fig. 5.5b,c. Alternatively, if the data changes sufficiently for the eigenvectors to change considerably, the projection shows a *rotation*-like artifact – see the two timesteps in Fig. 5.5d,e. In contrast to t-SNE and UMAP, these artifacts are not

due to stochastic seeding, but due to the way PCA works. Given the above, it is now clear why G-PCA is very stable – it chooses the two largest-variation axes for the *entire* dataset (all timesteps). The price to pay for this stability is that G-PCA may not yield the axes that best describe the data variation at each timestep, thus not the best spatial quality.

Autoencoders: Similarly to G-PCA, these techniques are stable since they train with the entire dataset (all timesteps) to learn a latent representation that encodes the global data distribution. Once trained, the encoder is a deterministic function that maps *nD* data to 2D. The main disadvantage of autoencoders over G-PCA is usability: PCA is simple to implement and use. Autoencoders, in contrast, have the 'usual' deep learning challenges, most notably finding the optimal network architecture and hyperparameter values.

5.5.2 *Finding similarly behaving techniques*

Figure 5.1 showed a high-level aggregated view of the quality metrics of the studied DR techniques, outlining that the autoencoders and PCA variants score better, in general, on both spatial quality and stability, than graph neighborhood techniques (t-SNE, dt-SNE, and UMAP). However, that image (and related analysis) was too aggregated. At the other extreme, Fig. 5.2 and related discussion showed a fine-grained analysis of all metrics measured for all techniques run on all datasets. From both these analyses, it is quite hard to understand how (and when) different techniques behave similarly. This is arguably important for practitioners interested in choosing a technique in a given context (dataset type and metrics to maximize).

Figure 5.6 supports this similarity analysis, as follows. Each point is here a technique run on a dataset, attributed by the computed 12 quality metrics. We project these points to 2D using UMAP, thus, creating a 'projection of projections' map. The four images in Fig. 5.6 use different visual codings to reveal several insights, as follows. Image (a) shows the techniques and datasets, coded by glyph, respectively categorical colors. Points in this plot are clustered more due to *datasets* than *techniques* – that is, quality is more driven by the dataset nature than by which projection technique is used. For instance, we see the *sorts* dataset well-separated as the purple cluster bottom-left in Fig. 5.6a. Images (b-d) show the same projection, colored by stability, distance preservation, and neighborhood preservation, respectively. The left part of the projection (orange dashed line, Fig. 5.6b) shows cases where stability and distance (and/or neighborhood) preservation are mutually complementary, *i.e.*, when we obtain high stability, we get low distance/neighborhood preservation and conversely. The top-right part of the projection (red dashed line, Fig. 5.6b) shows cases where both stability and spatial

quality are quite high. All these cases use the AE, VAE, and G-PCA techniques. The central area of the projection is covered mainly by t-SNE, dt-SNE and UMAP, telling that these projections have average behavior (as compared to autoencoders and PCA variants). Looking at the color-coded plots (images b-d), we see that these projections do not score highest on any of the considered metrics.

The plots in Fig. 5.6 can guide choosing a DR technique to project dynamic data: Given a dataset $\mathbf{D}$ to project, (1) find the most similar dataset $\mathbf{D}'$ in the benchmark, *i.e.*, that contains data of similar nature (*e.g.*, natural images, sounds) and is obtained via a similar acquisition process; (2) decide what is important for the dynamic projection of $\mathbf{D}$ – stability, distance preservation, neighborhood preservation, or a mix of them; (3) find the projection techniques P in the respective quality plots that have the desired qualities on $\mathbf{D}'$, and possibly also consider other projection techniques that behave similarly (close points in the plots). These techniques P are then good candidates to project $\mathbf{D}$ with.

5.6 CONCLUSION

This chapter is an initial step towards understanding the behavior of dimensionality reduction techniques in the context of dynamic/temporal data. We hope that the information and results presented here help practitioners who want to understand their complex data and that this work can be used by authors interested in developing DR techniques as a tool for evaluation and comparison. We proposed a publicly available benchmark with 9 methods, 10 datasets, and 12 quality metrics. To evaluate the viability of different techniques for the task, we computed spatial and temporal stability metrics for all possible combinations, thus providing an extensive collection of results. Based on the results, we presented a discussion that elaborates on the causes for understanding the dynamic behavior. All our experiments are documented and detailed online (The Authors, 2019) to allow further analysis and reproducibility.

There are many ways this work can be extended in the future. The benchmark can be extended with new methods, a better way to choose hyperparameters, new datasets, and new metrics. With a larger number of datasets, we can perform a robust test of the impact of dataset traits on the metrics. We can also integrate streaming data techniques, datasets, and tests. In this sense, it is definitely interesting to consider a more principled sampling of the 'universe' of all dynamic high-dimensional datasets, in the same way we considered the sampling of the universe of dynamic weighted hierarchies for the treemap evaluation in Chapter 3. This is a challenging endeavor, since we need to define relevant traits that describe different classes (types) of such datasets, as well as their dynamics. An equally challenging issue is finding representatives (samples) for these dataset classes. For treemaps, while difficult, we managed to find the relevant dynamic hierarchies, as there is a wide

offer of such hierarchies from various data sources. For dynamic multidimensional datasets, we found it comparatively far harder to locate such datasets in the public domain. As such, forming a good impression of what are relevant traits for these datasets, and next collecting relevant samples to evaluate dynamic projections, is an endeavor that we leave for future work.

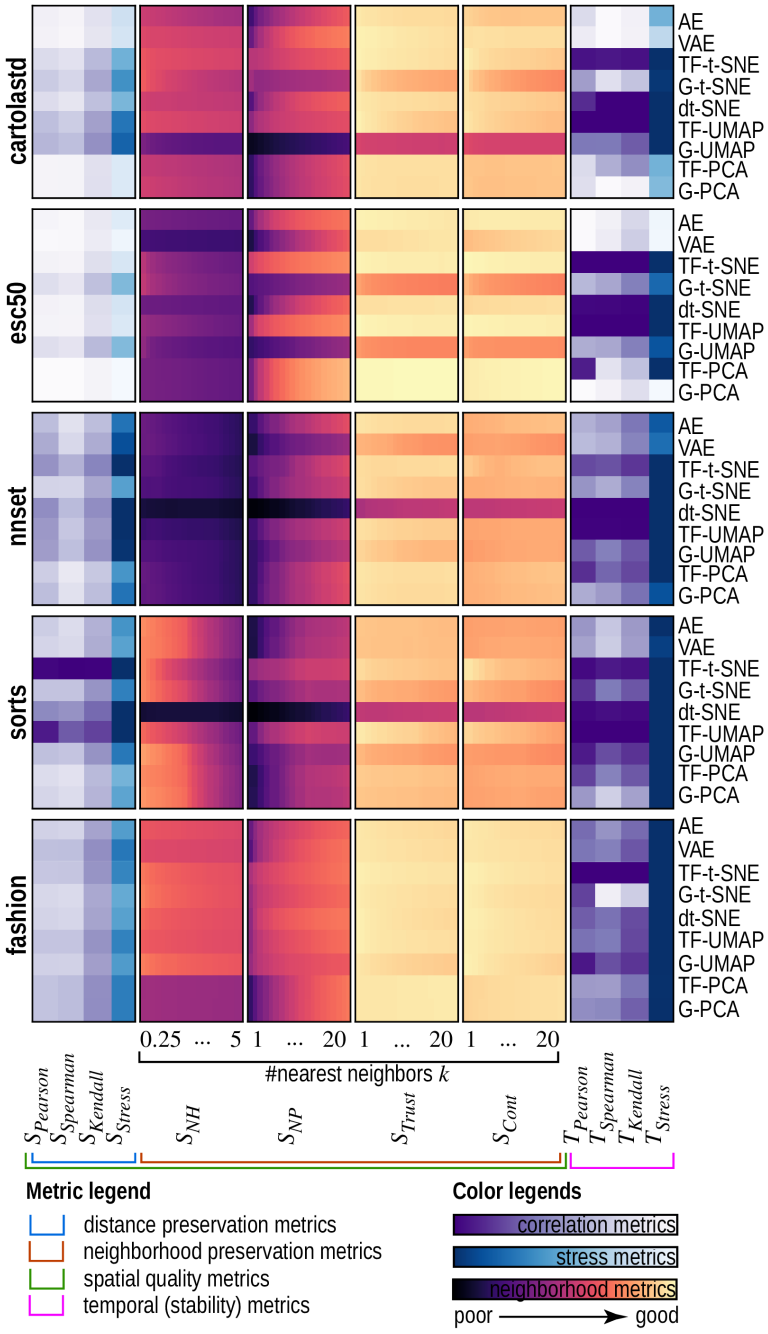


Figure 5.2: Twelve spatial quality and temporal stability metrics evaluated for 9 DR methods run on ten datasets.

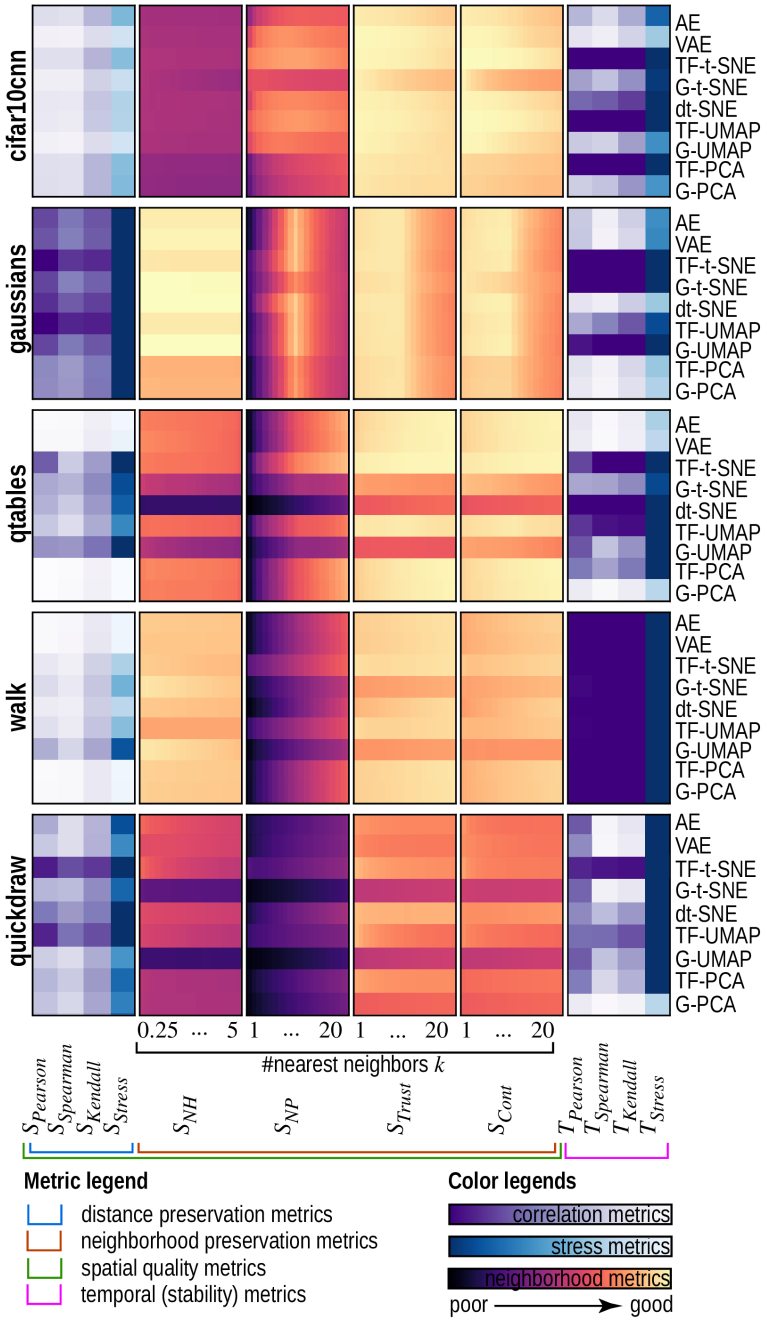


Figure 5.3: Twelve spatial quality and temporal stability metrics evaluated for 9 DR methods run on ten datasets.

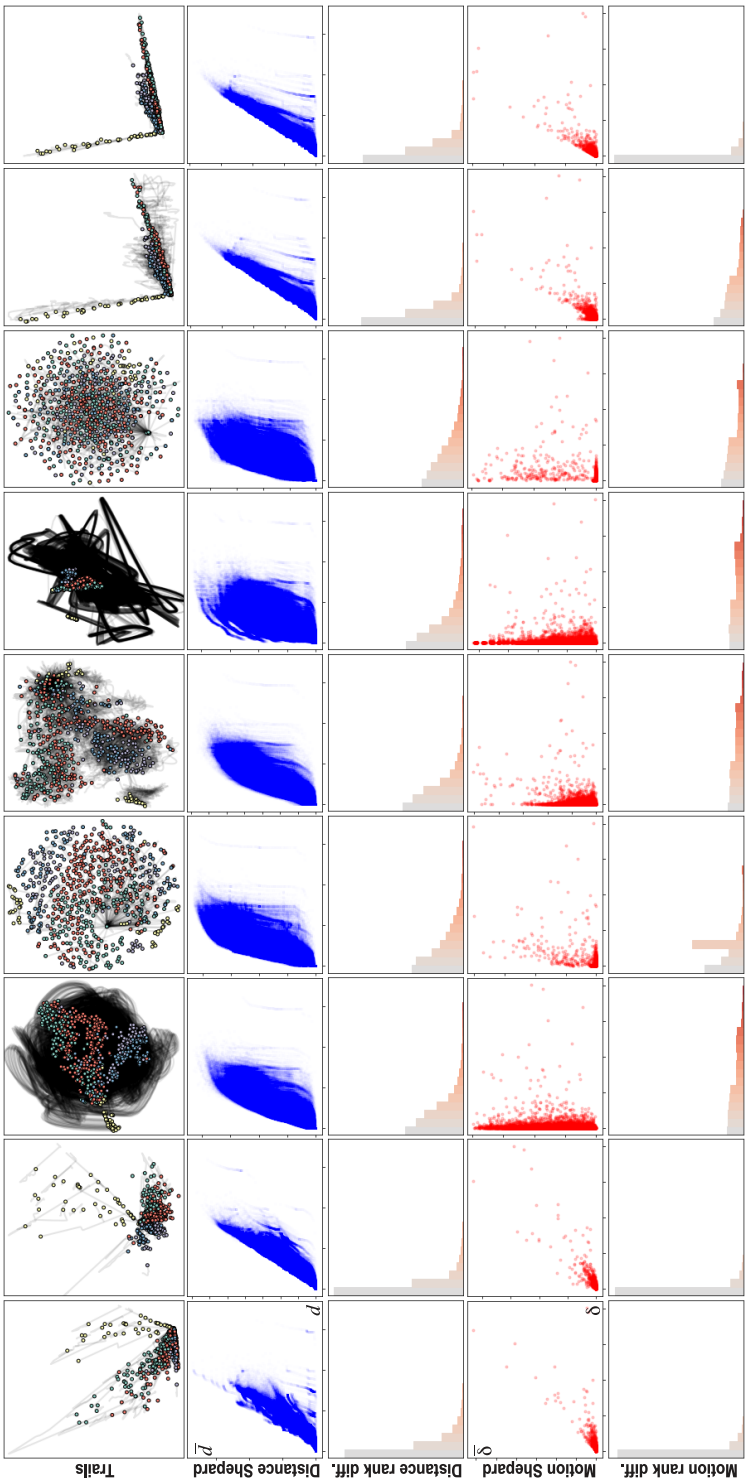


Figure 5.4: Detailed analysis of distances and movements produced by all DR techniques on the *cartolastd* dataset.

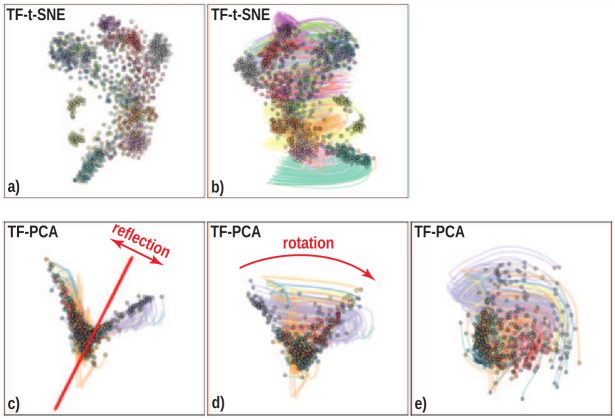


Figure 5.5: Examples of instability in TF-t-SNE (a,b) and TF-PCA (c,d,e).

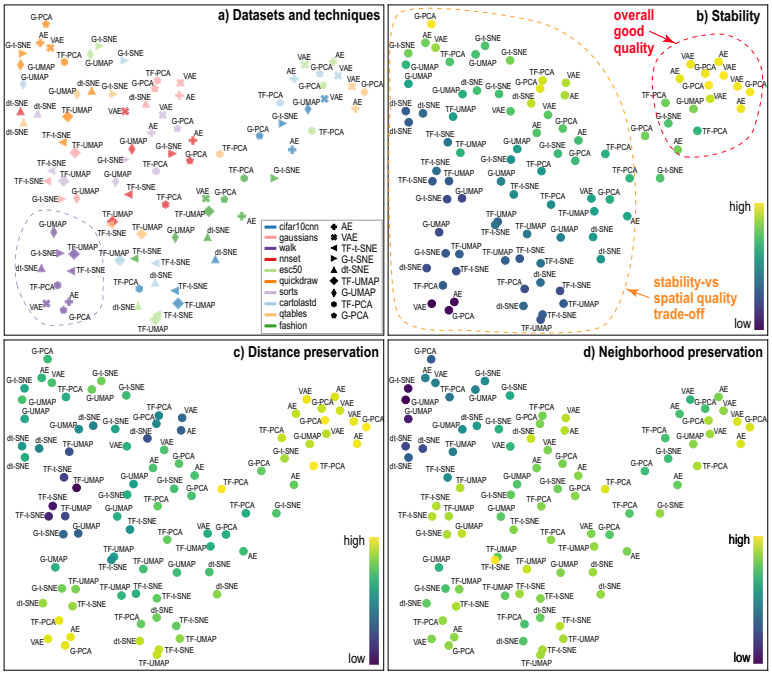


Figure 5.6: Projection of projections map showing the similarity of all evaluated techniques on all datasets (Sec. 5.5.2).

6 GUIDED STABLE DYNAMIC PROJECTIONS

In Chapter 5, we evaluated dynamic projections from the dual perspective of visual quality and stability, just as we did for dynamic treemap algorithms in Chapters 2-3. And, just as for dynamic treemap algorithms, we found that there is no winning dynamic projection method that scores best for both visual quality and stability. In this chapter, similar to our earlier work in Chapter 4 to improve dynamic treemaps, we aim to improve dynamic projections from the perspective of the two aforementioned metrics. For this, we propose two new dynamic projection algorithms: PCD-tSNE and LD-tSNE. The former uses information given by the Principal Components of the data over all timesteps to stabilize tSNE. The latter makes use of landmarks that guide points through stable trajectories. We compare these two new algorithms with existing methods for dynamic projection, and show their advantages, following the benchmark proposed in Chapter 5.

Abstract: Projections aim to convey the relationships and similarity of high-dimensional data in a low-dimensional representation. Most such techniques are designed for static data. When used for time-dependent data, they usually fail to create a stable and suitable low dimensional representation. We propose two dynamic projection methods (PCD-tSNE and LD-tSNE) that use global guides to steer projection points. This avoids unstable movement that does not encode data dynamics while keeping t-SNE's neighborhood preservation ability. PCD-tSNE scores a good balance between stability, neighborhood preservation, and distance preservation, while LD-tSNE allows creating stable and customizable projections. We compare our methods to 11 other techniques using quality metrics and datasets provided by a recent benchmark for dynamic projections.

6.1 INTRODUCTION

Many domains produce datasets with large numbers of observations (also called samples or points) and attributes (also called measurements, dimensions, or variables). Dimensionality reduction techniques, also called projections, are an established tool for visualizing such datasets in a simplified, compact, and scalable way.

This chapter is based on the paper "Guided Stable Dynamic Projections" (Vernier et al., 2021)

The literature on static projections – that address the visualization of time-independent datasets – is quite rich, with many techniques, surveys, and benchmarks on the subject (Nonato and Aupetit, 2019; Espadoto et al., 2019; Sorzano et al., 2014; Cunningham and Ghahramani, 2015). In contrast, far fewer techniques and comparisons thereof exist for projecting time-dependent datasets, in which the sample values change over time – which is a much harder problem.

Besides faithfully capturing the data structure – a desiderate shared with static projections – dynamic projections also face the challenge of maintaining temporal coherence. Failing this will create false motion artifacts in the projection, which can mislead the user into thinking there are data changes where none exist, or conversely. Figure 6.1 illustrates this: We have a 100-dimensional dataset of 2000 samples covering 10 distinct isotropic Gaussian distributions that collapse into 10 single points over 10 timesteps (Rauber et al., 2016). The images depict the results of three dynamic projection techniques (G-PCA, TF-PCA (Jolliffe, 1986), and TF-tSNE(van der Maaten and Hinton, 2008)) for this dataset, showing the trajectories of all data points over the ten timesteps. Knowing the dataset, we can tell that G-PCA renders quite faithfully the data dynamics and structure; TF-PCA creates an artificial amount of spiraling; and TF-tSNE creates a very large amount of apparently random and unstable motion that is not present in the data. If such variability in the projection results is seen for this simple, synthetic dataset, the choice of a good dynamic projection method for real-world datasets is clearly very hard.

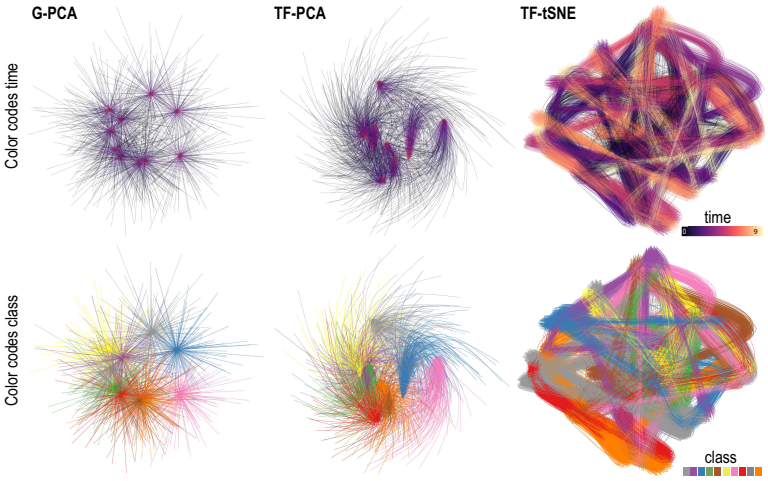


Figure 6.1: A time-dependent collapsing 100-dimensional 10-Gaussian-distributions dataset (2000 points) is visualized by three projection methods. Point trails are colored by time (top) and class (bottom). The images show increasing amounts of instability artefacts.

Motivated by these challenges of understanding and quantifying the quality of dynamic projections, Vernier et al. (2020a) evaluated nine such techniques, and came to the conclusion that there is no perfect method, and that an inherent trade-off between stability and spatial quality (i.e., neighborhood and distance preservation) exists. The methods that scored the best on both criteria were autoencoder-based methods and Global PCA. Neighborhood-based methods, such as t-SNE and UMAP, strongly showed a lack of stability. At the same time, these are among the favorite methods for static projection, given their high capability in preserving data structure.

We aim to cover the above-identified gap by proposing ways to add stability to the neighborhood preservation ability of static projections, in particular t-SNE. We propose two approaches that use global influences to steer projected points: Our first method, LD-tSNE, offers similar flexibility in steering dynamic projections via landmarks as known for static projections, and also reaches good quality values. Our second method, PCD-tSNE, increases neighborhood influences atop an already stable Global PCA dynamic projection, scoring better than all compared counterparts in terms of spatial quality combined with stability. The global influence of both methods can be controlled via simple user parameters to find the best balance between stability and spatial quality. We compare our methods with 11 existing dynamic projections on a benchmark of 10 high-dimensional datasets using 12 metrics for both spatial quality and stability.

Section 6.2 overviews related work on dynamic projections of high-dimensional data. Section 6.3 introduces our two new methods. Section 6.4 presents the experimental setup we used in comparing our new methods with existing ones. Section 6.5 presents and discusses the evaluation results. Finally, Section 6.6 concludes the paper.

6.2 RELATED WORK

6.2.1 *Preliminaries*

We first introduce some notations. Let $\mathbf{x} \in \mathbb{R}^n$ be an n-dimensional sample (also called data point or observation). A timestep $\mathbf{D}^t = \{\mathbf{x}_i^t\}$ of our data consists of a set of N samples $\mathbf{x}_i^t$, $1 \leq i \leq N$, measured at the same time moment t. A dynamic dataset $\mathbf{D}$ is a list of T timesteps $\mathbf{D} = (\mathbf{D}^t), 1 \leq t \leq T$. For simplicity of exposition and implementation, but without loss of generality, we consider next that the sample count N is constant over time. In this case, $\mathbf{D}$ can be represented as a set of T N-by-n matrices, one per timestep t.

A projection technique is a function $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$, where $q \ll n$. For visualization purposes, $q \in \{2, 3\}$. Since 2D projections are by far the most commonly used, we next only consider the case $q = 2$. We denote the projection of sample $\mathbf{x}$ by $P(\mathbf{x})$. For a timestep t, let

$P(\mathbf{D}^t) = \{P(\mathbf{x}^t)|\mathbf{x}^t \in \mathbf{D}^t\}$ be the 2D scatterplot of the projections of all points in $\mathbf{D}^t$. Finally, let $P(\mathbf{D})$ be the set of T scatterplots for all timesteps of dataset $\mathbf{D}$. These can be rendered as animations (see additional material (The Authors, 2020b)), trail sets (as in Fig. 6.1), small multiples (Rauber et al., 2016), or other visual encodings.

6.2.2 *Visualization of high-dimensional data*

Visualization of static high dimensional data (Liu et al., 2017) is a well studied topic populated with many techniques such as parallel coordinate plots (Inselberg and Dimsdale, 1990), table lenses (Rao and Card, 1994), scatterplot matrices (Becker et al., 1996), and dimensionality reduction (DR) methods or projections (Van Der Maaten et al., 2009). Compared to other methods, projections scale visually very well, being able to accommodate datasets of millions of samples and hundreds up to thousands of dimensions in limited screen space. Several quality metrics have been proposed to gauge how faithfully projections capture the structure of high-dimensional data, *e.g.*, trustworthiness and continuity (Venna and Kaski, 2006), normalized stress and Shepard diagrams (Joia et al., 2011), neighborhood hit (Paulovich et al., 2008), class consistency (Tatu et al., 2010), and distance consistency (Sips et al., 2009). Tens of different projection algorithms exist for static data; detailed taxonomies of such methods, benchmarks, and qualitative and quantitative evaluations are available in a range of surveys (Nonato and Aupetit, 2019; Espadoto et al., 2019; Fodor, 2002; Cunningham and Ghahramani, 2015; Sorzano et al., 2014; Van Der Maaten et al., 2009).

6.2.3 *Strategies for dynamic projections*

All current dynamic projection techniques that we are aware of are based on methods that were initially designed for static data. These base methods are adapted to achieve two goals: (a) obtaining good *spatial quality*, measured by the various static projection metrics outlined earlier in Sec. 6.2.2; and (b) obtaining good *stability*, defined as the ratio between changes, over time, of the projection $P(\mathbf{D})$ *vs* changes of the data $\mathbf{D}$ (Vernier et al., 2020a). Besides projections, similar definitions of stability have been used to quantify dynamic treemapping algorithms (Vernier et al., 2020b, 2018b). We next propose to classify these techniques as a function of how they 'adapt' the underlying base (static) projection algorithm, denoted further P_B, to handle spatial quality and stability for dynamic data.

Per-timeframe (TF): In this simplest strategy, P_B is applied to each timestep $\mathbf{D}^t$ to create an independent projection $P_B(\mathbf{D}^t)$. Hence, $P(\mathbf{D}) = (P_B(\mathbf{D}^t))_{1 \leq t \leq T}$. In other words, the base method P_B is not

allowed to "look at the past or future" when projecting a given timestep t – it only sees the data in $\mathbf{D}^t$. Given the popularity of PCA (Jolliffe, 1986), t-SNE (van der Maaten and Hinton, 2008), and UMAP (McInnes et al., 2018), the per-timeframe strategy is often used for these base projections, leading to variants we call next TF-PCA, TF-tSNE, and TF-UMAP, respectively. Several further variations of this strategy exist. Bach et al. (2016) propose time curves which connect consecutive positions $P_B(\mathbf{x}_i^t)$ of the same point i for all moments t, using MDS for P_B. Similar curves have been used by Bernard et al. (2012) (using PCA for P_B). Brich et al. (2020) use time curves and argue for the pro's and con's of PCA *vs* MDS for P_B. However, none of the studied base projections was found ideal concerning stability and spatial quality. At a more general level, the same strategy was used to connect different 2D scatterplots created by other means than projections (Haroz et al., 2015). Jäckle et al. (2016) use MDS for P_B to project all n spatial dimensions of $\mathbf{D}$ to a single dimension and use the second dimension of the screen space to map time. Overall, the per-timeframe strategy favors spatial quality, which can be as high as delivered by P_B. However, stability can be (very) low since P_B is applied independently to the timeframes.

Global (G): At the other end of the spectrum, global methods apply P_B to the entire dataset, and then separate the projected points based on their timesteps, *i.e.*, $P(\mathbf{D}) = (\{\mathbf{y}^t \in P_B(\mathbf{D})\})_{1 \leq t \leq T}$, where $\mathbf{y}^t = P_B(\mathbf{x}^t)$ and $\mathbf{x}^t \in \mathbf{D}^t$. Like per-timeframe, this strategy is also simple to implement. It maximizes stability by construction. As such, many applications use this strategy, *e.g.* Hu et al. (2010) that project 72-dimensional human body keypoints using LLE, or Fujiwara et al. (2018) who project entire dimensions (time series) using MDS and t-SNE for computer performance analysis. The latter method was also extended to use PCA and UMAP as P_B (Fujiwara et al., 2020).

When $\mathbf{D}$ is large, either in terms of number of samples or number of timesteps, computing a single projection $P_B(\mathbf{D})$ can be expensive. Also, the spatial quality of global techniques is typically lower than for the per-timeframe strategy since P_B now has to optimize the relative placement of points in *all* timeframes, even if such points never co-exist at the same time. Out-of-sample projection (OOS) methods can help with these issues. Simply put, an OOS technique P is constructed to optimize the projection of a subset $\mathbf{D}_s \subset \mathbf{D}$ according to one's desired quality metrics. Next, P is used to extrapolate the projection to the entire $\mathbf{D}$. Out-of-sample strategies have been proposed for many static projection methods (Bengio et al., 2003). Recently, Espadoto et al. (2020b) have shown how to use deep learning to construct out-of-sample approximations of any static projection technique. Hence, OOS techniques can be used to accelerate and potentially increase the quality of global projection methods. However, the challenge is in

how to select the small subset $\mathbf{D}_s$ so as to represent well the entire time-dependent dataset $\mathbf{D}$. To our knowledge, no studies of this aspect exist for dynamic projections.

Continuous (C): This strategy applies to base methods that iteratively optimize neighborhood configurations, such as t-SNE and UMAP. In the following, we call these variants C-tSNE and C-UMAP, respectively. The projection $P(\mathbf{D}^t)$ continues the gradient descent from the positions of the previous timestep $P(\mathbf{D}^{t-1})$, with the updated cost function for t. This reduces significantly the non-deterministic behavior created by removing consecutive initialization steps. Still, this can fail to produce stable projections as points are still allowed to move significantly during optimization. Dynamic t-SNE (D-tSNE) (Rauber et al., 2016) aims to alleviate this by adding a penalty term to the continuous strategy using t-SNE for P_B. This limits, up to a certain extent, too large point movements between consecutive timesteps. Incremental PCA (Ross et al., 2008) projects points in a streaming fashion and is therefore amenable to project time-dependent data. Fujiwara et al. (2020) further increase incremental PCA's stability by using Procrustes analysis to align consecutive projections, a method also proposed independently by Joia et al. (2011). Neves et al. (2020) propose Xtreaming, an incremental technique that handles streaming high-dimensional data by continuously adapting UPDis (Neves et al., 2018), a projection with out-of-sample capability, thus, good stability. Overall, continuous strategies achieve a good balance between spatial quality and stability. However, this balance can be hard to tune in practice.

Vernier *et al.*'s evaluation (Vernier et al., 2020a) found that PCA and (Variational) Autoencoders with the global strategy – called next G-PCA, G-VAE, and G-AE respectively – were the best-suited methods for projecting temporal data. The global strategy, however, does not seem to work well with graph or neighborhood-based methods, such as t-SNE and UMAP – we denote these methods next as G-tSNE and G-UMAP, respectively.

6.3 GUIDED METHODS FOR DYNAMIC PROJECTION

Many guided methods exist in the static projection literature (Nonato and Aupetit, 2019; Sorzano et al., 2014). Simply put, all these methods select a subset of samples $\mathbf{L} \subset \mathbf{D}$ to create P, by extrapolating $P(\mathbf{L})$ to $P(\mathbf{D})$. Conceptually speaking, the continuous strategy (Sec. 6.2.3) can be seen as a type of guidance, where $P(\mathbf{D}^{t+1})$ is steered by the earlier projection $P(\mathbf{D}^t)$. Similarly, the out-of-sample global strategy (Sec. 6.2.3) can be seen as a type of guidance where $P(\mathbf{D}_s)$ steers $P(\mathbf{D})$. However, even though this works for simple datasets, when the data present complex dynamics and large changes over time, existing continuous strategies become too restrictive. We propose two new guided

methods for dynamic projection that use global influences (landmarks or suggested placements) to steer and stabilize the projection while still accounting for neighborhood preservation. The two methods use t-SNE as base projection given (a) t-SNE's high popularity for the static projection case; and (b) the difficulty of using t-SNE in a dynamic context (see Sec. 6.2.3), which we want to overcome. Importantly, while guided strategies mainly aim to address *scalability* for static projections, our different aim of using guidance is to address *spatial quality* and *stability*.

6.3.1 *Landmark Dynamic t-SNE (LD-tSNE)*

One idea that has been successfully used in the static case, and can be utilized to our advantage for dynamic data, is the use of *landmarks*. Landmarks or similar control point-based mechanisms are well known and have been used to aid different tasks on static data. Examples include performance improvement(Pekalska et al., 1999; Silva and Tenenbaum, 2003; De Silva and Tenenbaum, 2004; Vladymyrov and Carreira-Perpiñán, 2013; Paulovich et al., 2008; Kruiger et al., 2017a), support of out-of-sample capability(Boytsov et al., 2017; Poličar et al., 2019), and projection customization(Joia et al., 2011; Neves et al., 2018). Yet, we are not aware of any work that combines landmarks or control points to stabilize *dynamic* projections. We use landmarks to give the base projection P_B method a sense of global structure, in an attempt to reduce the instability inherent to neighborhood-based projection techniques such as t-SNE.

Two main aspects must be considered when using landmarks as guides: how to generate the landmarks and how to use the landmarks to steer points, as follows.

Landmark generation: Each landmark $\mathbf{l} = (\mathbf{l}^n, \mathbf{l}^q)$ consists of a high-dimensional component $\mathbf{l}^n \in \mathbb{R}^n$ and a component $\mathbf{l}^q \in \mathbb{R}^q$ in the projection space. It is important that the set $\mathbf{L} = \{l^n\}$ captures well the structure of the high-dimensional dataset $\mathbf{D}$, otherwise the "steering" may become uneven. There are different ways of achieving this goal (De Silva and Tenenbaum, 2005). For simplicity and speed, we opted to create $\mathbf{L}$ by randomly sampling k points from $\mathbf{D}$, where k is a fraction of the size of $\mathbf{D}$. For most of our tests, we set $k = N$, *i.e.*, the number of points in a timeframe (see Appendix in supplementary material). To generate the low-dimensional points $\mathbf{l}^q$, we simply project $\mathbf{L}$ using a user-chosen method. We experimented here with both PCA and t-SNE, and selected the landmark projection which yielded the best results (see Appendix in supplementary material).

Landmark steering: The first step towards steering is to select a neighborhood-based projection technique to use. We chose here t-SNE

due to its popularity and previous good results in extending it for dynamic data (Rauber et al., 2016). To describe how steering takes place, let us consider the original t-SNE cost function, given by the Kullback-Leibler (KL) divergence between the joint-probability distributions $\mathcal{P}$ and $\mathcal{Q}$ that describe point-neighborhoods in $\mathbb{R}^n$, respectively $\mathbb{R}^q$

$$C_{tsne} = D_{KL}\left(\mathcal{P}||\mathcal{Q}\right) = \sum_{i=1}^{N}\sum_{j\neq i} p_{ij}\log\frac{p_{ij}}{q_{ij}}, \tag{6.1}$$

where $p_{ij} = \frac{p_{i|j}+p_{j|i}}{2N}$ models the distance of two points $\mathbf{x}_i$ and $\mathbf{x}_j$ in $\mathbb{R}^n$ and

$$p_{j|i} = \frac{\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2/\left(2\sigma_i^2\right)\right)}{\sum_{k\neq i}^{N}\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2/\left(2\sigma_i^2\right)\right)}.$$

Here, $p_{j|i}$ can be seen as a relative measure of similarity based on the local neighborhood of a point $\mathbf{x}_i$. The effective number of neighbors considered for each point is given indirectly by a user-chosen perplexity value μ: The value of σ_i is computed so that, for the user-given μ and each i, $\mu = 2^{-\sum_j^N p_{j|i}\log_2 p_{j|i}}$.

A Student's t-distribution with one degree of freedom is used to compute the joint-probability distribution in $\mathbb{R}^q$ as

$$q_{ij} = \frac{\left(1+\left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}}{\sum_{k,i\neq k}\left(1+\left\|\mathbf{y}_i - \mathbf{y}_k\right\|^2\right)^{-1}}.$$

The gradient of the cost function, given by

$$\frac{\partial C_{tsne}}{\partial \mathbf{y}_i} = 4\sum_j\left(p_{ij}-q_{ij}\right)\left(\mathbf{y}_i-\mathbf{y}_j\right)\left(1+\left\|\mathbf{y}_i-\mathbf{y}_j\right\|^2\right)^{-1} \tag{6.2}$$

is used to incrementally move the points $\mathbf{y}_i$ to reduce the cost C_{tsne}.

To add landmark influence to t-SNE we will, similarly to Rauber et al. (2016), add a second term to the cost function. In their work, the extra term was used to penalize *any* kind of 2D movement. In our case, we want to *guide* the placement of points $\mathbf{y}_i$ based on the similarity of $\mathbf{x}_i$ with the landmarks $\mathbf{l}^n$. Figure 6.2 illustrates these global and local influences. In Fig. 6.2a, the landmarks in $\mathbf{L}$ (light blue) produce attraction and repulsion forces to guide the placement of the red point $\mathbf{y}_i$. In Fig. 6.2b, the remaining points $\mathbf{y}_j$, $j \neq i$ (gray in the figure), exert similar forces, influencing and being influenced by $\mathbf{y}_i$, just like in a regular t-SNE projection.

We weigh the global and local influences by a factor $\lambda \in [0,1]$ giving the total cost function

$$C = (1-\lambda)C_{tsne} + \lambda C_{landmarks}. \tag{6.3}$$

In the above, $C_{landmarks}$ is similar to the original t-SNE cost function C_{tsne}. However, instead of considering p_{ij} for all pairs of points in $\mathbf{D}$ or $\mathbf{D}^t$, we let only the landmarks $\mathbf{l} \in L$ act upon each $\mathbf{y}_i$, *i.e.*

$$C_{landmarks} = \sum_i \sum_{l \in L} p_{i|l} \log \frac{p_{i|l}}{q_{il}}. \tag{6.4}$$

For these influences to work consistently through all time steps t, several aspects differ from the original t-SNE. In Eqn. 6.4, we use the *asymmetric* $p_{i|l}$ instead of the symmetric p_{il} used in Eqn. 6.1. Indeed, we want the landmarks to influence the points, not the other way round. Secondly, for the computation of σ_l for each landmark, we only take into consideration the landmark points $\mathbf{L}$. These two modifications ensure that the forces are consistent and do not fluctuate depending on the local density of points in $\mathbf{D}$ or $\mathbf{D}^t$.

From Eqns. 6.2, 6.3, and 6.4, we find the gradient of C as

$$\begin{aligned}\frac{\partial C}{\partial \mathbf{y}_i} = (1-\lambda)\left(4\sum_j \left(p_{ij} - q_{ij}\right)\left(\mathbf{y}_i - \mathbf{y}_j\right)\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}\right) \\ +\lambda\left(4\sum_{l \in L} \left(p_{i|l} - q_{ij}\right)\left(\mathbf{y}_i - \mathbf{y}_j\right)\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}\right).\end{aligned}$$

To accelerate convergence, improve initialization, and create tighter clusters, exaggeration terms are used (van der Maaten and Hinton, 2008; van der Maaten, 2015; Linderman et al., 2019; Linderman and Steinerberger, 2017). These are scalars that multiply p_{ij}, suggesting greater similarity between points than $\mathcal{P}$ captures. We do the same by adding two factors α and β to grant additional influence on how much points in $\mathbf{D}^t$ affect each other (α = local), respectively how much the landmarks "pull" the projected points (β = global), leading to the final cost gradient

$$\begin{aligned}\frac{\partial C}{\partial \mathbf{y}_i} = (1-\lambda)\left(4\sum_j \left(\alpha p_{ij} - q_{ij}\right)\left(\mathbf{y}_i - \mathbf{y}_j\right)\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}\right) \\ +\lambda\left(4\sum_{l \in L} \left(\beta p_{i|l} - q_{ij}\right)\left(\mathbf{y}_i - \mathbf{y}_j\right)\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}\right).\end{aligned}$$

Regarding algorithmic complexity, the original unoptimized implementation of the t-SNE method is $\mathcal{O}(n^2)$ for both computation and memory (van der Maaten and Hinton, 2008). Our LD-tSNE algorithm has an additional cost $\mathcal{O}(ln)$ given by the interaction of the landmarks with the points in the projection, where l is the number of landmarks and n the number of points in the projection. Therefore, the final time and memory complexity are given as $\mathcal{O}(n^2 + ln)$, or, since n^2 dominates the cost, LD-tSNE can be considered $\mathcal{O}(n^2)$.

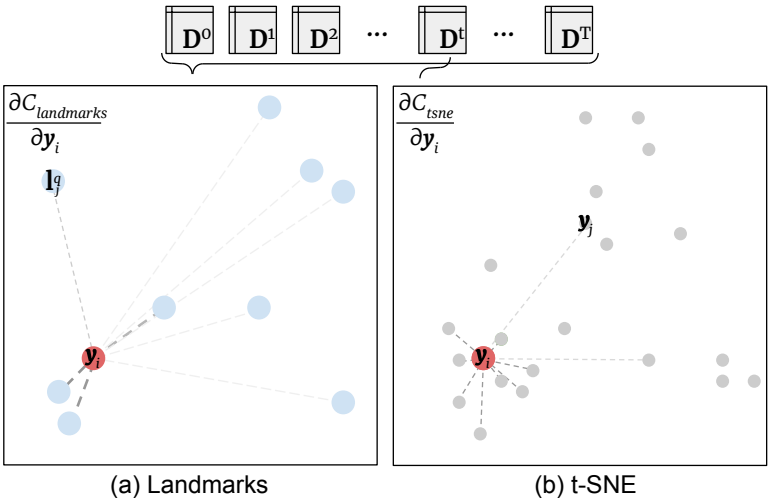


Figure 6.2: Effect of landmarks (a) and regular projection points (b) upon a point $\mathbf{y}_i$ in LD-tSNE. See Sec. 6.3.1.

6.3.2 *Principal Component Dynamic t-SNE (PCD-tSNE)*

Our second dynamic projection, PCD-tSNE, is a guided method that allies the stability of G-PCA with the neighborhood preservation capabilities of t-SNE. Just like D-tSNE and LD-tSNE, it includes an additional term to the t-SNE cost function that adds stabilization to the otherwise unstable C-tSNE.

The first step in PCD-tSNE is to compute a projection matrix W constructed from the top-q eigenvectors of the covariance matrix of $\mathbf{D}$. Simply put, W describes the (two, in our case) orthogonal axes of largest data variation over the whole dataset. For each point $\mathbf{x}_i \in \mathbf{D}$, we apply a transformation $\mathbf{x}_i W$ to map $\mathbf{x}_i$ to $\mathbb{R}^q$. More specifically, this places $\mathbf{x}_i$ exactly as G-PCA would, which was proven earlier (Vernier et al., 2020a) to create *stable* projections.

The placement of each projection point $\mathbf{y}_i$ is next given by two factors (see Fig. 6.3): an attraction to the position $\mathbf{x}_i W$, marked in light blue in Fig. 6.3a; and the influence of all other points in $\mathbf{D}^t$ upon $\mathbf{y}_i$, as given by tSNE, these points being shown in gray in Fig. 6.3b. With these elements, the gradient of our cost function is given by:

$$\frac{\partial C}{\partial \mathbf{y}_i} = (1-\lambda)\frac{\partial C_{tsne}}{\partial \mathbf{y}_i} + \lambda \|\mathbf{y}_i - \mathbf{x}_i W\|. \tag{6.5}$$

Here, $\lambda \in [0,1]$, similarly to LD-tSNE, weighs the balance of local and global influences. More specifically, by adjusting λ, we can achieve an exact C-tSNE projection ($\lambda = 0$), an exact G-PCA projection ($\lambda = 1$), or a projection in between these variants.

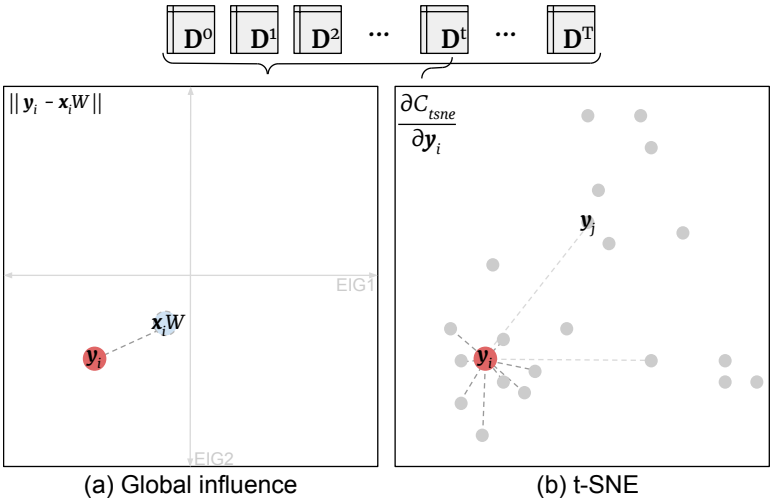


Figure 6.3: Effect of global influence (a) and regular projection points (b) upon a point $\mathbf{y}_i$ in PCD-tSNE. See Sec. 6.3.2.

Regarding complexity, for PCD-tSNE, the first step is to compute the top-2 eigenvectors of the original data. There are many numerical methods designed to efficiently perform this computation with cost as low as $\mathcal{O}(kn)$ for k singular values (Kressner, 2005; Cline and Dhillon, 2006). Once the PCD-tSNE optimization starts, the additional term introduced on Equation 6.5 represents a simple convex function, which means that convergence is reached efficiently and PCD-tSNE performs similarly to C-tSNE, that is, in $\mathcal{O}(n^2)$ time.

6.4 EVALUATION PROCEDURE

We next present our evaluation of the two proposed dynamic projection methods, LD-tSNE and PCD-tSNE. For the evaluation, we started with the benchmark in Vernier et al. (2020a), which is to our knowledge the only benchmark dedicated specifically to dynamic projections. The implementation of per-timeframe and global methods were provided by Vernier et al. (2020a); D-tSNE was provided by Rauber et al. (2016); the remaining techniques considered in our evaluation (Sec. 6.3) were implemented by ourselves. All source code, datasets, and obtained results can be found in our online repository (The Authors, 2020b).

6.4.1 *Methods*

The Vernier et al. (2020a) benchmark (described in the previous chapter) contained 9 methods – five global ones (G-AE, G-VAE, G-tSNE, G-UMAP, G-PCA), three per-timeframe ones (TF-tSNE, TF-UMAP, TF-

PCA), and one continuous (D-tSNE). Atop of those we added C-tSNE and C-UMAP, and the two newly proposed methods, LD-tSNE and PCD-tSNE. The parameters used in the benchmark are available in the supplemental material.

6.4.2 *Quality Metrics*

Following Vernier et al. (2020a) and Espadoto et al. (2019), we used 8 *spatial* and 4 *temporal* quality metrics, as follows. Temporal metrics measure the correspondence of movement of projection points in $\mathbb{R}^q$ with regard to their change in the data space $\mathbb{R}^n$ space, *i.e.*, stability.

6.4.2.1 *Spatial metrics*

Spatial metrics measure how well a projection maps the underlying high-dimensional data, and can be divided into neighborhood preservation metrics ($S_{NP}, S_{NH}, S_{Trust}, S_{Cont}$) and distance preservation metrics ($S_{Stress}, S_{Pearson}, S_{Spearman}, S_{Kendall}$). Note that these do not necessarily relate to how humans perceive the projection (Wang et al., 2018).

Neighborhood preservation (S_{NP}) is the fraction of the k-nearest neighbors of $\mathbf{x} \in \mathbf{D}$ whose projections are in the k-nearest neighbors of $P(\mathbf{x})$.

Trustworthiness (S_{Trust}) measures how well the k nearest neighbors $\nu^k(P(\mathbf{x}))$ of a projected point $P(\mathbf{x})$ match the k nearest neighbors $\nu^k(\mathbf{x})$ of a data point $\mathbf{x}$, specifically, how *few* missing neighbors (Martins et al., 2014) a projected point has. If $U^k(\mathbf{x})$ is the set of points in $\mathbf{D}$ that project in $\nu^k(P(\mathbf{x}))$ but are not in $\nu^k(\mathbf{x})$, and $r^P(\mathbf{x}, \mathbf{y})$ is the rank of $\mathbf{y}$ in the ordered set of nearest neighbors $\nu^k(P(\mathbf{x}))$, trustworthiness is defined as $1 - \frac{2}{Nk(2N-3k-1)} \sum_{x=1}^{N} \sum_{y \in U^k(\mathbf{x})} (r^P(\mathbf{x}, \mathbf{y}) - k)$.

Continuity (S_{Cont}) measures how *many* missing neighbors a projected point has. Let $V^k(\mathbf{x})$ be the points that are in $\nu^k(\mathbf{x})$ but do not project in $\nu^k(P(\mathbf{x}))$. Let $r(\mathbf{x}, \mathbf{y})$ be the rank of $\mathbf{y}$ in the ordered set of neighbors $\nu^k(\mathbf{x})$. Continuity is then defined as $1 - \frac{2}{Nk(2N-3k-1)} \sum_{x=1}^{N} \sum_{y \in V^k(\mathbf{x})} (r(\mathbf{x}, \mathbf{y}) - k)$.

Neighborhood hit (S_{NH}) is the fraction of the k-nearest neighbors of a projected point $P(\mathbf{x})$ that have the same class label as $P(\mathbf{x})$. Since we use labeled datasets with reasonably well-separated classes in $\mathbb{R}^n$ (see next Sec. 6.4.3), a projection P that is good for class-separation tasks should have a high S_{NH} value.

All the above metrics range in $[0, 1]$, with 1 indicating optimal value. We compute S_{NP}, S_{Trust}, and S_{Cont} for multiple (20) neighborhood

sizes equally spread between $k = 1\%$ and $k = 20\%$ of the point count N. For S_{NH}, we use 20 values for k, ranging from 0.25% to 5% of N. We next average the results for different neighborhood sizes k, following (Vernier et al., 2020a; Martins et al., 2015).

Normalized stress (S_{Stress}) measures the pairwise difference of distances of points in $\mathbf{D}$ and $P(\mathbf{D})$. We define S_{Stress} as $\sum_t \sum_{ij}(d_{ij}^t - \delta_{ij}^t)^2 / T \sum_{ij}(\delta_{ij}^t)^2$, where d_{ij}^t and δ_{ij}^t are the Euclidean distances between data points $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$, and between their projections $P(\mathbf{x}_i^t)$ and $P(\mathbf{x}_j^t)$, respectively for every point pair (i, j) and timeframe $1 \leq t \leq T$. To ease analysis, we scale distances using standardization.

Shepard diagram metrics. The Shepard diagram is a scatterplot of d_{ij} by δ_{ij}, for every point pair (i, j) (Joia et al., 2011). A diagram close to a diagonal line indicates good distance preservation. Scatterplots spreading above or below the diagonal indicate distance compression (potential false neighbors), respectively stretching (potential missing neighbors) from $\mathbf{D}$ to $P(\mathbf{D})$. We use Pearson correlation, Spearman rank, and Kendall tau to measure the linearity and monotonicity of the relationship of d_{ij} with δ_{ij} in Shepard diagrams. The three resulting metrics $S_{Pearson}, S_{Spearman}, S_{Kendall}$ range in $[-1, 1]$, with 1 being the ideal distance-preservation case.

6.4.2.2 *Temporal stability metrics*

We estimate how stable a projection is by studying the relationship of the *data* change of a point from $\mathbf{x}_i^t$ to $\mathbf{x}_i^{t+1}$, measured by $c_i^t = \|\mathbf{x}_i^t - \mathbf{x}_i^{t+1}\|$, and the movement of the corresponding *projections* from $P(\mathbf{x}_i^t)$ to $P(\mathbf{x}_i^{t+1})$, measured by $\kappa^t = \|P(\mathbf{x}_i^t) - P(\mathbf{x}_i^{t+1})\|$. For stable P, we ideally would want κ_i^t to be proportional, or at least correlated with, c_i^t. We use the following metrics (Vernier et al., 2020a) to capture this notion of stability.

Normalized temporal stress (T_{Stress}) is defined as $\sum_{i\,t}(c_i^t - \kappa_i^t)^2 / (c_i^t)^2$. As for S_{Stress}, we normalize distances using standardization. Low T_{Stress} values tell that the $\mathbb{R}^q$ changes κ_i^t reflect closely their $\mathbb{R}^n$ counterparts c_i^t, which is what we want.

Temporal Shepard diagram metrics: We measure the Pearson and Spearman correlation and Kendall's tau ($T_{Pearson}, T_{Spearman}, T_{Kendall}$) between c_i^t and κ_i^t for every sample i and timestep t. High values indicate that the $\mathbb{R}^q$ changes κ_i^t are strongly correlated with their $\mathbb{R}^n$ counterparts c_i^t, which is desirable.

6.4.3 *Datasets*

We used 10 public datasets extracted from different sources and portraying a wide range of temporal phenomena, such as videos, sound recordings, sports statistics, algorithm behavior, and a few synthetic datasets with easily recognizable dynamics (The Authors, 2020b). The collection also exhibits significant variations in measurable traits such as the number of samples N, the number of timesteps T, dimensionality n, intrinsic dimensionality ρ_n (percentage of dimensions that describe 95% of the data variance), and sparsity ratio σ_n (percentage of zeros in the data), as shown by Table 6. These traits have been used earlier (Espadoto et al., 2019) to indicate that a benchmark captures an as wide as possible (within the benchmark's size bounds) spread of phenomena of different natures.

cartolastd: This dataset has player statistics for the second turn of the 2017 Brazilian soccer championship. Data was extracted from an open-source project (Gomide and Gualberto, 2019) that scrapes the Cartola FC (Gomide and Gualberto, 2019) platform. Each of the 19 timesteps is a tournament round. Samples are players, with dimensions being per-match performance (number of goals, assistances, fouls, defenses) and player position (goalkeeper, right or left-back, defender, midfield, forward).

cifar10cnn: Samples are images classified by a convolutional network trained for the CIFAR10 (Krizhevsky, 2009) dataset. Dimensions are activations of neurons of the network's last hidden layers. Timesteps represent training epochs. This dataset is similar to the one produced for MNIST (LeCun and Cortes, 2010) by Rauber et al. (2017b), but consider the significantly harder-to-classify CIFAR10 dataset.

esc50: Sound samples of 8 classes (brushing teeth, chainsaw, crying baby, engine, laughing, rain, siren, wind) compressed to 128 frequencies and smoothed over time. Extracted from Piczak's ESC50 dataset (Piczak, 2015).

fashion: This is a subsample of 100 images from each of the 10 classes (T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot) of the FashionMNIST (Xiao et al., 2017) dataset. Each image is recorded over 10 timesteps, with decreasing amounts of noise over time.

gaussians: Isotropic gaussian blobs in $\mathbb{R}^{100}$ with diminishing spread over time. Used originally to evaluate D-tSNE (Rauber et al., 2016).

nnset: Internal states (weights and biases) of several neural networks during 30 training epochs to learn classifying the MNIST dataset. The

Table 6: Datasets used and their traits (from Vernier et al. (2020a)).

dataset	samples N	timesteps T	dimensions n	classes	intrinsic dim. ρ_n	sparsity ratio σ_n
cartolastd	696	19	17	5	0.6470	0.0000
cifar10cnn	1000	30	10	10	0.6599	0.0000
esc50	320	108	128	8	0.0345	0.0000
fashion	1000	10	784	10	0.4762	0.2971
gaussians	2000	10	100	10	0.3680	0.0000
nnset	80	30	8070	8	0.0057	0.0001
qtables	180	40	1200	9	0.0077	0.0007
quickdraw	600	89	784	6	0.4309	0.9013
sorts	80	100	100	8	0.3505	0.0100
walk	300	50	100	3	0.4783	0.0001

networks have the same architecture but use different optimizers, batch sizes, and training-set sizes.

qtables: Internal state of agents learning to move a car up a hill using the reinforcement learning algorithm Q-learning (Watkins, 1989). The nine classes represent variations of learning rates and discounts.

quickdraw: Drawing sequences for 600 objects of 6 different classes drawn by random people. Samples represent the pixels of individual drawings. Timesteps (89) represent the drawing stage. Data is extracted from the "Quick, Draw!" Google AI experiment (Jongejan et al., 2016).

sorts: This dataset was designed to compare eight sorting algorithms. They sort each different arrays of 100 random values in $[0, 1]$. We take snapshots of the algorithms' intermediate states until sorting completion. A sample is an (algorithm, array) run, its dimensions being the partially-sorted array values at a given sorting step.

walk: Synthetic dataset similar to *gaussians* (Rauber et al., 2016), but with more complex dynamics. It contains 3 high-dimensional clusters that oscillate (approach, mingle, cross, and then drift apart) in $\mathbb{R}^{100}$ over 50 timesteps. This dataset tests how well the studied projections can capture the approaching, mingling, and drifting-away dynamics mentioned above.

6.5 EVALUATION RESULTS

We used each of the selected 13 projection techniques (Sec. 6.4.1) to project the 10 datasets in the benchmark (Sec. 6.4.3). For every (dataset, method) pair, we compute 12 quality metrics (4 related to distance preservation, 4 related to neighborhood preservation, and 4 stability met-

rics, see Sec. 6.4.2), and analyze the results at different levels of aggregation. For a direct impression, the animations of each (dataset, method) pair can be found in our online repository (The Authors, 2020b).

6.5.1 *Visual comparison of dynamic projections*

We start with a simple, visual comparison of dynamic projection results. Figure 6.4 shows the trail-sets – curves linking $P(\mathbf{x}_i^t)$ for all t – for the *cifar10cnn* dataset, created by the 13 tested dynamic projection methods, organized following the taxonomy in Sec. 6.2.3. Points represent the last layer of neural network activations trained to classify the dataset, over 30 training epochs. Given the problem, we *expect* that activations 'segregate' into 10 distinct sets, corresponding to the 10 classes of images in the dataset. The trails should start from a roughly common area (middle of the projection), indicating lack of differentiation at training start, and evolve *smoothy*, that is, without major twists and bends, over epochs, to increasingly differentiated clusters, a phenomenon shown earlier in for the (far) easier-to-classify MNIST dataset (Rauber et al., 2017b) by C-tSNE. We see that only a few dynamic projections exhibit this pattern: G-VAE, G-UMAP, and our proposals, PCD-tSNE and LD-tSNE. All other dynamic projections do not show the convergence of trails to (ten) distinct clusters (red circles in the figure). Saliently, all TF variants show far too high dynamics - long trails turning and twisting, suggesting chaotic dynamics, which we know it is not the case from Rauber et al. (2017b). Other projections (G-AE, G-PCA, G-tSNE, C-UMAP, D-tSNE) do not show a clear convergence of trails into 10 clusters, which again, we know should be expected. Overall, we argue that PCD-tSNE and LD-tSNE capture the (known) ground-truth of the training dynamics better than most tested counterparts.

6.5.2 *Overview of quality metrics*

Figure 6.6 shows the results for each method separated by metric class. The three swarm plots (Eklund, 2012) in the figure address each of the three metric categories outlined above (distance preservation, neighborhood preservation, temporal stability). Columns in a plot indicate methods. Each point in a column corresponds to the averaged result over the four normalized metrics in the respective class for a (method, dataset) pair. Methods in each plot are ordered by how high they score for a given metric class, with methods to the left scoring higher. Methods are categorically color-coded to ease comparison between the plots.

A method to be considered suitable for dynamic projections must be stable and achieve good distance and neighborhood preservation. Vernier *et al.*'s benchmark (Vernier et al., 2020a) concluded that, from all their 9 tested methods, G-PCA and Autoencoder-based techniques

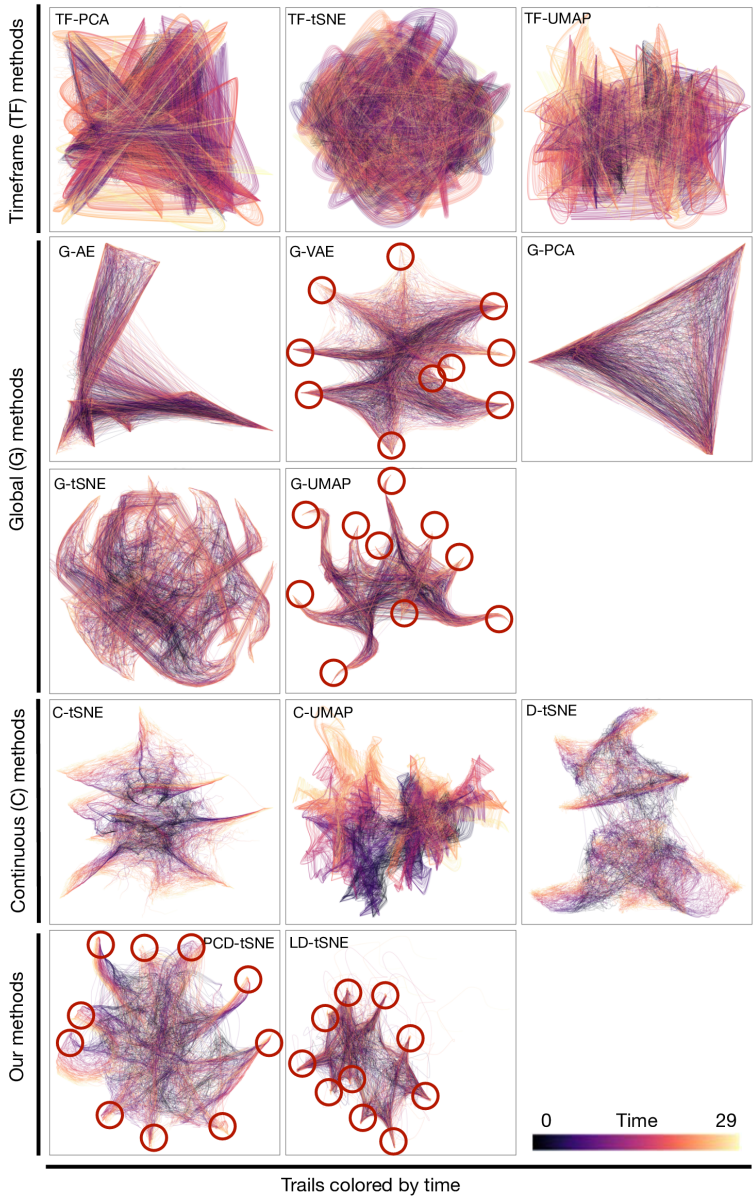


Figure 6.4: Trails showing the "hidden activity" (Rauber et al., 2017b) of a convolutional neural network trained on the *CIFAR10* (Krizhevsky, 2009) dataset, computed by all 13 tested dynamic projections. Red circles show clusters of trail endpoints which indicate training convergence. Images without red circles show (suboptimal) projection methods where it is not possible to see this training convergence.

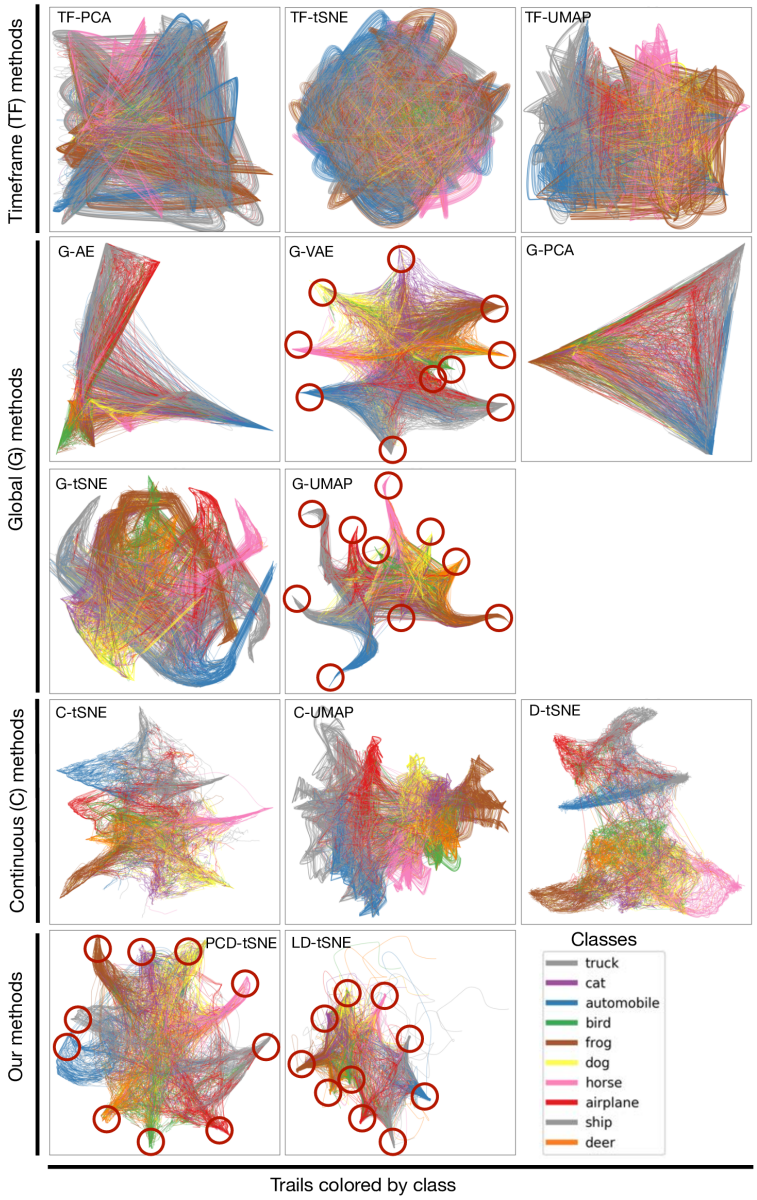


Figure 6.5: Identical trails to Fig. 6.4 but colored by data class.

(G-AE, G-VAE) struck the best balance between these desiderates. We argue that, in this light, our new PCD-tSNE method is even more effective at projecting dynamic data. Indeed, as Fig. 6.6 (bottom plot) shows, PCD-tSNE's stability is comparable to G-PCA, G-AE, and G-VAE, being the third-most stable of all tested methods. At the same time, PCD-tSNE

achieves better results in distance preservation, being the best of all tested methods. Regarding neighborhood preservation (Fig. 6.6, middle plot), PCD-tSNE is only surpassed by the TF (timeframe) and C (continuous) methods. This is not surprising since these methods do not have temporal constraints. This implies, as Fig. 6.6 (bottom plot) confirms, that the TF and C methods score very poorly for stability.

Additionally, PCD-tSNE overcomes two limitations of AE-based methods and G-PCA: Autoencoders are based on neural networks, which can be challenging to set up, optimize for the architecture, and train; PCA based methods are sensitive to outliers and do not explicitly try to preserve local features.

Concerning LD-tSNE, we see that this method did not achieve metric results as good as other state-of-the-art methods. Yet, it scores in the top half of all methods for all three considered metric classes. Also, its strength lies in its customizability (see next Sec. 6.5.5): If we want the projection to adhere to a certain shape, or we have some prior knowledge over the high-dimensional space and we want areas of the projections to carry a certain data-related semantic, we can easily place landmarks to drive the projection to that behavior. This extends the same flexibility, known earlier for static projections (see *e.g.* (Joia et al., 2011; Pekalska et al., 1999)), to dynamic projections.

6.5.3 *Stability and spatial quality trade-off*

While the swarm plots in Fig. 6.6 help us see which methods score best for a given metric class, they do not let us easily compare methods from the perspective of *multiple* metrics. To achieve this, we use two star plots (Fig. 6.7), as follows. Each image is a scatterplot having temporal stability as the x axis and distance and neighborhood preservation, respectively, as the y axis. Each colored point shows the average metric values for a given technique over all datasets. Spokes emerging from a point show the average metric values for each of the 10 datasets run by the respective technique. For more insight into the behavior of the methods, we highlighted the spokes for the two best methods in each plot, *i.e.* the points placed closest to the top-right corner of the plot.

Figure 6.7a shows that PCD-tSNE and G-VAE are the best methods for distance preservation *and* stability, closely followed by G-PCA. Yet, the spokes of PCD-tSNE (pink) are shorter than those of G-VAE (blue). That is, PCD-tSNE achieves a consistently higher distance preservation and stability over all 10 tested datasets than G-VAE, which has a higher variability. Similarly, Figure 6.7b shows that PCD-tSNE scores highest in terms of neighborhood preservation *and* stability, closely followed by G-VAE and G-AE. Again, the spokes of PCD-tSNE are shorter than those of G-VAE, telling that PCD-tSNE achieves its high scores more consistently than G-VAE. We see that G-VAE performs worst for the *walk*, *nnset*, and *fashion* datasets, from both the perspective of distance

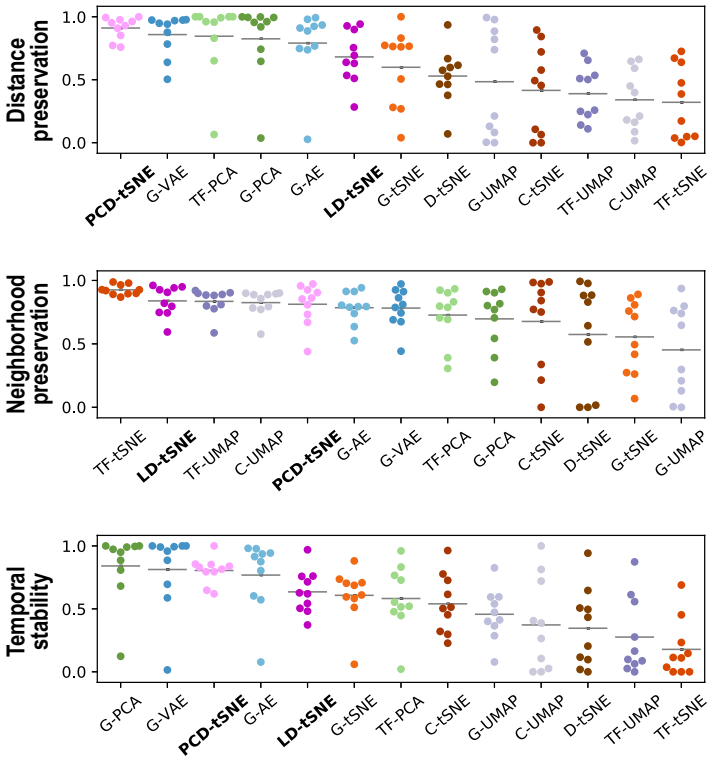


Figure 6.6: Swarm plot ordering methods from best to worse for each metric class. Each point corresponds to the average metric result over the 4 metrics in a given class normalized to $[0, 1]$ for each (method, dataset) pair. Horizontal lines show average metric values over all datasets for each (method, metric class) pair.

preservation and neighborhood preservation (longest blue spokes in Figs. 6.7a,b for G-VAE, indicated by a cross, triangle, and check icons); for these datasets, PCD-tSNE performs quite well (short pink spokes). Also, there seems to be an inverse correlation between neighborhood and distance preservation for TF-tSNE, TF-UMAP, and G-UMAP, indicating that these methods are very good at neighborhood, but not distance, preservation. Separately, we see that our two methods, PCD-tSNE and LD-tSNE, are the best methods, stability-wise, from the t-SNE class, and perform far better, on all three metrics, than D-tSNE. In other words, if one wants to leverage t-SNE's ability for dynamic datasets, our methods are the best from the considered variants. Finally, we see that temporal stability and distance preservation appear to be well correlated over all tested methods (points in Fig. 6.7a close to the diagonal), which is to our knowledge a new finding in the projection literature. In contrast, no similar correlation appears between stability and neighborhood preservation (Fig. 6.7b).

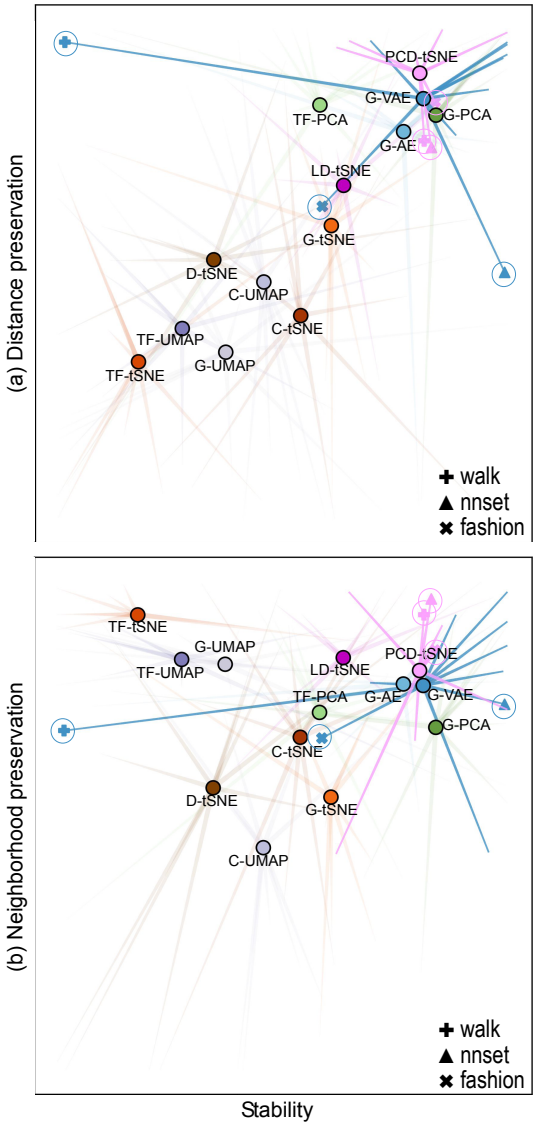


Figure 6.7: Star plots compare dynamic projection methods from the perspective of stability *vs* distance (a) and stability *vs* neighborhood (b) preservation. A point shows the average values of these metrics for a given technique over all datasets. Spokes show the average metric values for each dataset for a given technique. Spokes of PCD-tSNE and VAE, the two techniques that score best, are in bold. Methods with big spoke fans show high variation on quality metrics. Short spokes show consistent results of a method over all 10 datasets.

6.5.4 *Global* vs *local influence control*

As outlined in Sec. 6.3.2, the PCD-tSNE method has a parameter λ that modulates the amount of global influence applied to the points being

projected. When projecting a sample $\mathbf{x}_i \in \mathbf{D}$, this global influence refers to minimizing the distance of $P(\mathbf{x}_i)$ to the position given by the transformation matrix W composed of the top-q eigenvectors of $\mathbf{D}$. Another way to interpret this global influence is to think of $\mathbf{x}_i W$ as the position that G-PCA would generate; and to think about λ as how much we want PCD-tSNE to approximate G-PCA.

If we use high λ values (close to 1), PCD-tSNE gets very close to G-PCA, a method that has shown to be very stable, produce good distance preservation, but has low neighborhood preservation (Fig. 6.6). Conversely, with low λ values (close to 0), no global influences act upon PCD-tSNE, which turns into C-tSNE, a method that has high neighborhood preservation, but low stability and distance preservation (Fig. 6.6). Figure 6.8 supports and refines this insight. For each dataset (rows), we compute the mean distance preservation (MDP), mean neighborhood preservation (MNP), and mean temporal stability (MTS) over the respective metrics in each class (see Sec. 6.4.2). For each table row, we normalize values between 0 and 1, to better see the spread of values of the respective metric for each dataset. The leftmost column shows the metric results of G-PCA; the rightmost one shows the results of C-tSNE. The six middle columns show the results of PCD-tSNE with λ in $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$. Cells are colored using an ordinal colormap (dark green=low, bright yellow=high metric values) on the normalized values. The color gradients show that PCD-tSNE indeed yields metric values that are very similar, for high, respectively low, λ to those of G-PCA, respectively C-tSNE. Also, we see that PCD-tSNE can integrate characteristics of both C-tSNE and G-PCA and achieves the best balance between all quality metrics, as shown by the overall brightest columns in the middle of the table. Interestingly, PCD-tSNE is also often able to achieve the best result for certain quality metrics (maximum averages, marked bold in Fig. 6.8). This shows that PCD-tSNE doesn't simply *interpolate* projections (like, for example, in Kruiger et al. (2017b)), but uses the characteristics of both C-tSNE and G-PCA to create a better projection. In Fig. 6.8, note that different rows show different trends, which is expected since we consider different datasets and metrics.

Finally, Figure 6.8 shows that G-PCA and C-tSNE are not always optimal – the best projection lies sometimes in between, which is what PCD-tSNE obtains. Separately, it shows that optimal parameters depend on the dataset. The considered MDP, MNP, and MTS quality metrics could be used for automatic finding of such optimal parameters – or good preset values for all datasets – by grid search, following the approach in Espadoto et al. (2019) for static projections.

6.5.5 *Using landmarks to steer dynamic projections*

The key trait of LD-tSNE is that it allows steering a *dynamic* projection by changing the landmark point positions $\mathbf{l}^q$. If we monitor a high-

dimensional process and we know what final and failed states look like, we can place landmarks indicating these states. As the process evolves, we will have a clear picture of which samples failed or succeeded; which samples are on the "right track"; and how similar samples are in in-between states among themselves and to these known states. Guiding landmarks are also valuable if we want a system that is consistent over slightly different datasets.

Landmark-based methods present many challenges in practice: How to choose a small set of points in $\mathbb{R}^n$ that is representative of $\mathbf{D}$? How many points do we need, and how do we select these representatives? De Silva and Tenenbaum (2005) propose regression models for picking the best landmarks, while Pezzotti et al. (2016) use a hierarchical approach. Another option is to synthesize landmarks using models that approximate the manifold (e.g. Autoencoders). For simplicity, we select our landmarks by random sampling $\mathbf{D}$, as in earlier work considering static projections, *e.g.* (Joia et al., 2011; Pekalska et al., 1999).

Figure 6.9 shows how landmark placement can steer a dynamic projection. We use the *gaussians* dataset, as we know its dynamics, so we can assess how well landmark steering works on the resulting projections. Points are colored per cluster; landmarks are drawn white. Figure 6.9a shows several timesteps of LD-tSNE with landmarks placed by G-PCA. We see how clusters 'implode' over time. While clusters stay roughly in the same place over time in the projection (a good indication of stability), their *spatial* organization is not ideal for monitoring the phenomenon. Figure 6.9b shows LD-tSNE for the same dataset, with the same landmarks $\mathbf{l}^n$ selected from $\mathbf{D}$, but with the 2D landmarks $\mathbf{l}^q$ placed manually into 10 horizontally-aligned, similar-size, clusters. The images show the same 'implosion' effect over time as in Fig. 6.9a. We argue that the *dynamics* of the data is now much easier to see due to the separation of clusters given by our 2D landmarks' placement. The point made is that the freedom of landmark placement of LD-tSNE allows one to separate the issues of spatial disentanglement of samples in the projection (done by the landmark placement) from monitoring the dynamics of the data (taken care of by LD-tSNE). The Appendix (supplementary material) shows additional information on this and related experiments.

6.6 CONCLUSION

We have presented two projection methods that leverage the good neighborhood-preservation ability of t-SNE for dynamic (time-dependent) data. For this, we use guidance in the form of landmarks (for our first method, LD-tSNE), respectively attractors to principal vectors (for our second method, PCD-tSNE). We compared our methods against 11 dynamic projection techniques on 10 datasets using 8 spatial quality and 4 stability metrics. The comparison showed that PCD-

tSNE scores better than all compared methods on the combined spatial quality and stability criteria. LD-tSNE obtained second-best scores on neighborhood preservation, allowing flexible placement of landmarks to drive the shape of the resulting dynamic projection. While our work – for sure – does not solve the problem of dynamic projection of high-dimensional data, we argue that our methods bring added value to users interested in this goal.

We next aim to extend our methods to handle streaming data. Adapting our work to use deep learning, similar to Espadoto et al. (2020b), would lead to high-quality and computationally scalable dynamic projections. Additional validation of our methods on more datasets, and with concrete use-cases and user tasks, is also important. Finally, developing new metrics to measure the quality of dynamic projections for specific tasks, thereby extending the insights in Nonato and Aupetit (2019) for the dynamic case, is a long-term goal we aim to pursue.

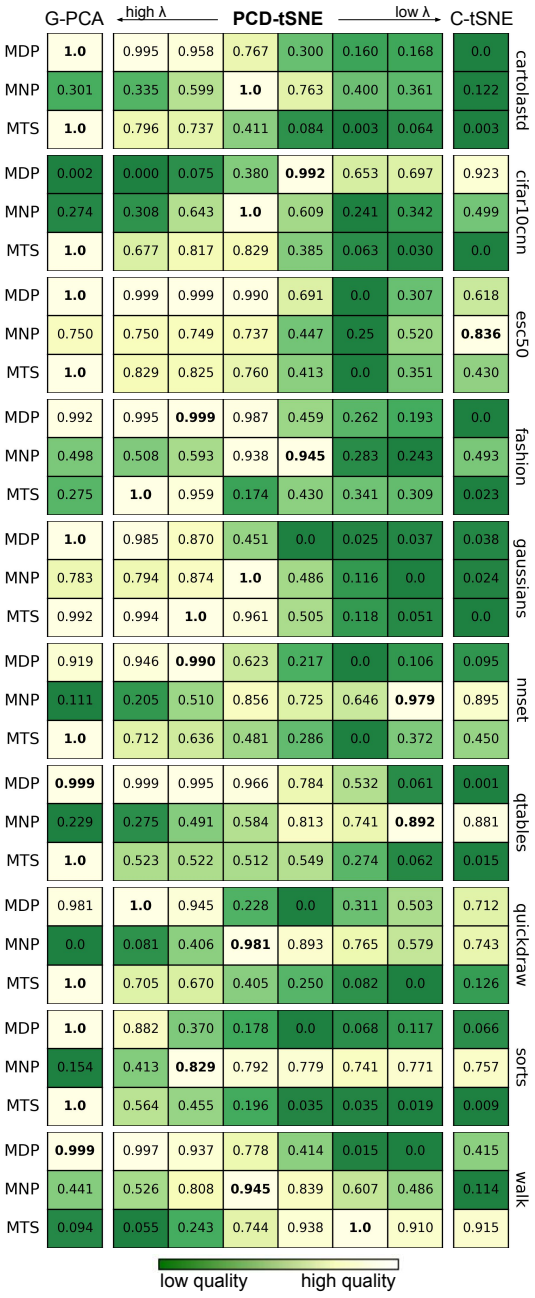


Figure 6.8: Mean distance preservation (MDP), mean neighborhood preservation (MNP), and mean temporal stability (MTS) per dataset, as in Figs. 6.6 and 6.7, but normalized over the 8 runs in each subplot. The leftmost column is for G-PCA. The next 6 columns are for PCD-tSNE with $\lambda \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$. The rightmost column is for C-tSNE. By changing λ, PCD-tSNE generates a smooth gradient, simulating G-PCA and C-tSNE at the extremes and producing hybrids in-between (Sec. 6.3.2). The best balance between all metric classes is often found in this compromise.

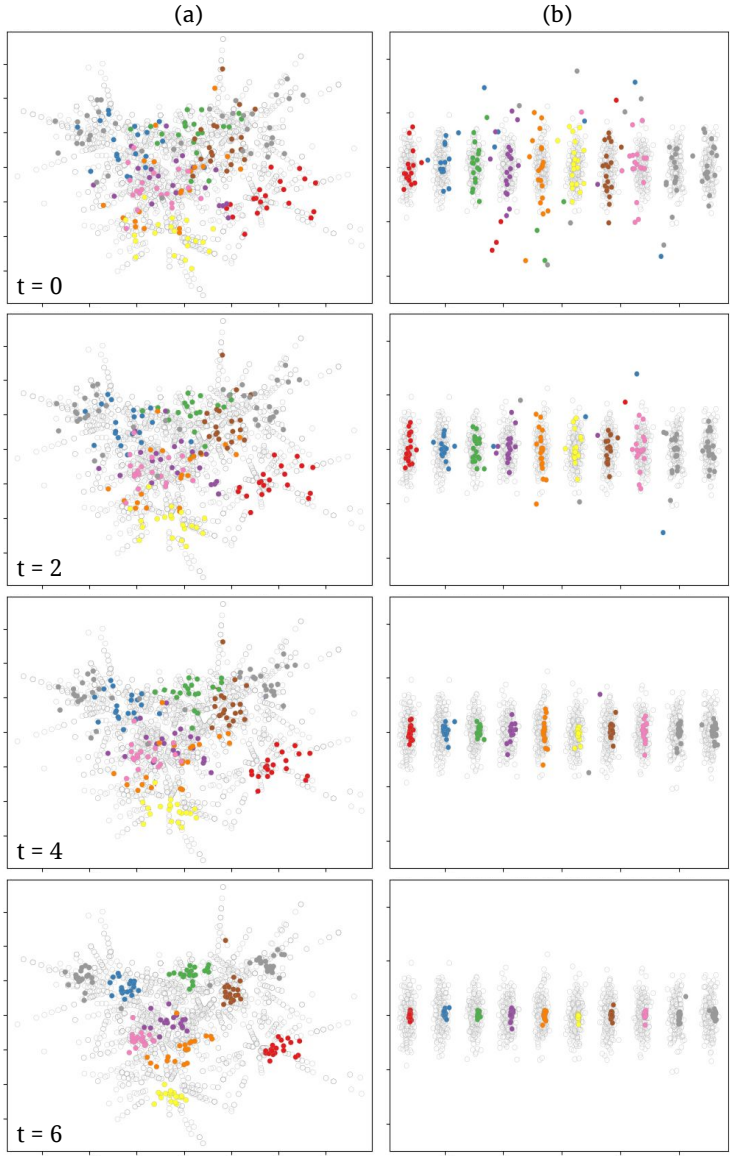


Figure 6.9: Projection of the *gaussians* dataset with landmarks (gray points) placed (a) by G-PCA and (b) manually according to cluster label. Points are colored by cluster label. The implosion dynamics known to be present in the data is visible in both cases. Yet, the manual landmark placement creates a less cluttered view.

7 HYPERKINETIC MOVEMENT DISORDER ANALYSIS

In this chapter, we present a real-world application for the dynamic projection methods introduced in Chapters 5 and 6. The context of our application is the analysis of hyperkinetic movement disorders. These disorders manifest as abnormal involuntary movements that highly affect the quality of life of the people who suffer from them. These involuntary movements may present a wide range of tendencies: they may be regular and rhythmic (tremors); swift "lightning-like" jerks or twitches (myoclonus); or sustained and repetitive movements resulting in abnormal postures (dystonia); they may present a random, brief, and non-rhythmic character (chorea); or can be temporarily suppressible jerks (tics). The diagnosis of these disorders is carried out via careful professional observation. It can be extra challenging due to the circumstantial emergence of certain behaviors (triggered by specific postures or tasks) and their manifestation via compound movements, including a combination of the various hyperkinesias. All these factors lead to professionals often disagreeing with diagnosis. This chapter focuses on electromyography (EMG) and motion sensors (accelerometry) data collected from patients with hyperkinetic movement disorders while performing motor tasks. We show how we transform these data and study them with a visual analytics tool based on dynamic projections. We discuss an example of data exploration that leads to valuable clinical insights.

7.1 INTRODUCTION

Hyperkinetic movement disorders are a class of disorders that are characterized by excessive and involuntary motor movements. These disorders include a range of different phenotypes[1] such as myoclonus, dystonia, tremor, chorea and tics. Since the clinical presentation and etiology differs among these disorders, each of these disorders requires a different clinical strategy. These strategies differ in e.g., the choice of additional diagnostic tests, medication prescriptions, and deep brain stimulation targets. For the correct strategy, it is of utmost importance to ensure accurate phenotypic classification. As most hyperkinetic movement disorders have no clear anatomical brain abnormalities, classification is based on clinical definitions and thus on expert opinion. What makes this increasingly difficult is that complex and mixed forms of phenotypes occur in many patients, and many clinical features of these

1 A phenotype is any observable characteristic or trait of a disease, such as morphology, development, biochemical or physiological properties, or behavior, without any implication of mechanism.

hyperkinetic movement disorders also correspond to clinical features of other disorders classes such as ataxia, spasticity, and functional movement disorders. Moreover, there is large inter- and intra-observer classification variability in expert clinicians (van der Veen et al., 2021; Defazio et al., 2004; Eggink et al., 2017; Beghi et al., 2014; van der Salm et al., 2013).

To improve hyperkinetic movement disorder phenotyping, the Next Move in Movement Disorders (NEMO) (van der Stouwe et al., 2021) study was set up at the University Medical Center Groningen, The Netherlands. The aim of this study is to build computer-aided hyperkinetic movement disorder classification tools to assist healthcare professionals in phenotyping (see further Sec. 7.3).

The current chapter focuses on the EMG and accelerometry data of the NEMO study. EMG is a technique that measures electrical activity in skeletal muscles. This can either be done by placing an electrode inside the muscle of interest, or on the skin above the muscle. The latter technique is referred to as surface EMG and is employed in the current study. The simplest application of EMG is to determine whether a muscle is active. Accelerometry data collects information on the 3D motion of several sensors placed on the body of a subject that performs a task.

The above EMG and accelerometry data contains a wealth of information describing the dynamics of a subject that performs a task involving motion. The underlying assumption we have, on which the NEMO study is also based, is that analyzing such data will allow us to find patterns specific to particular types of hyperkinetic movement disorders. In turn, this will help the design of computer-aided systems for disorder characterization and classification.

However, the joint EMG and acceletrometry data is large (in terms of the number of sample points), high-dimensional (has many independent variables), and temporal – and, as such, it is challenging to analyze and interpret. On the other hand, this is precisely the type of data for which our dynamic projection methods presented in the previous chapters were designed.

We explore the idea of using dimensionality reduction (DR) methods on the EMG and accelerometry data in order to support the exploration of the data collected in the NEMO experiments. We are not directly trying to solve the clinical problem of phenotypical classification of movement disorders. This is a much harder problem, the pursuit of which would encompass one thesis of its own. Our goal is to understand the technical challenges associated with applying DR methods to such complex time-dependent EMG and accelerometry data sets to support effective data exploration. We show evidence that the dynamic projection methods we proposed in earlier chapters are suitable to handle such complex data and generate interesting insights in this particular clinical setting, opening new possible paths of analysis, which were previously unavailable due to technical limitations.

7.2 RELATED WORK

In clinical practice, EMG is often combined with accelerometry, a method that measures the acceleration of limb displacement rather than muscle activity. Measurements are typically performed during rest or during diagnostic movement tasks and clinically assessed using several characteristics of the patient's movement pattern, such as muscle activation patterns, movement burst duration, and frequency analysis (van der Veen et al., 2021).

More advanced analyses to help support the classification of tremor, myoclonus, or dystonia focus on the electrophysiological autospectrum to investigate the frequency distribution using Fourier transforms, standard coherence analysis to investigate the dependence of multiple signals in the frequency domain, wavelet coherence analysis to investigate the variation of coherence in the time domain, cumulant density to investigate the relationship between signals in the time domain, and Jerk-locked back-averaging to investigate signal relationships during events of interest (*i.e.*, jerks) across measurement modalities, respectively (Nijmeijer et al., 2014; Tijssen et al., 2000; Grosse et al., 2004; Kramer et al., 2018; van der Stouwe et al., 2015; Grosse et al., 2003).

Unfortunately, there are several known limitations to the diagnostic value of EMG and accelerometry features in hyperkinetic movement disorder phenotyping, three of which we will discuss briefly. First, high-level evidence for the differentiating value of each feature is sparse. Only a limited number of diagnostic test accuracy studies exist, most of which report on tremor patients. Many of the features that are currently a part of the diagnostic criteria for movement disorders have only been reported in descriptive studies, and have not been compared between patient groups, *e.g.*, in myoclonus versus tremor patients. Hence, incorporating these features in clinical practice is largely based on clinical observations and expert opinion. Secondly, proper application of EMG and accelerometry techniques and interpretation of the results requires training and experience. Both practice and quality range widely between medical centers, and diagnostic accuracy are highest in specialized centers. Thirdly, the application of EMG and accelerometry features is complicated by the inherent nature of movement disorders themselves: As all of these disorders are defined by excessive involuntary movement, it is only to be expected that many movement disorder phenotypes share overlapping features. Combined, these three factors currently limit the diagnostic value of EMG and accelerometry features for movement disorders.

7.3 HYPERKINECTIC MOVEMENT DISORDERS AND EXPERIMENT DESIGN

The experimental design of the NEMO study is described in detail in (van der Stouwe et al., 2021). In short, a large data set is being collected from hyperkinetic movement disorder patients (20 dystonia, 20 myoclonus, and 20 tremor patients) and 40 healthy controls. In the future, the study aims to include additional patient groups. Importantly, disorder phenotype classification in this data set is based on independent expert panel agreement.

In the study, participants perform 36 motor tasks during a movement registration setting, and 1 motor and 3 non-motor tasks in neuroimaging settings. During movement registration, data is collected using electromyography (EMG), motion sensors (accelerometry), and 2D and 3D video. In the neuroimaging settings, data is measured using positron emission tomography (PET) and functional magnetic resonance imaging (fMRI).

Participants are only eligible for inclusion if they are at least 16 years old and healthy participants cannot be first-degree relatives of patients with hyperkinetic movement disorders.

The current chapter focuses on the data from the movement registration. During this setting, participants performed 36 tasks that are used in the clinical setting. Tasks were selected based on panel discussions with several movement disorders, neuropediatric, and neurorehabilitation specialists with extensive experience in the fields of dystonia, myoclonus, tremor, chorea, tics, ataxia or spasticity to ensure coverage of all disorders.

7.4 VISUAL ANALYSIS OF COLLECTED DATA

Our goal was to design a visual analytics tool that supports exploring the motion data generated in the NEMO project. This is important given that the data is vast and has many axes that can be explored. Exploring this large data corpus is challenging. Thus, we aim to provide medical professionals with a tool to *navigate* through and generate valuable *insights* from patient motion patterns effectively and efficiently. Our tool aims to support a wide range of capabilities, such as identifying clusters, outliers, erratic observations, or failures in the data collection. It also aims to support medical analysis tasks, such as comparing various hyperkinesias to healthy behavior or identifying under which circumstances certain abnormal tendencies become incident. In both cases, we must be able to compare the severity and variability of their manifestation.

Another essential aspect our tool aims to provide is *meta-analysis*, where we want to know which combinations of sensors, tasks, patient groups, and preprocessing transformations generate representa-

tive data that may be useful in further investigation steps. Such meta-analysis is helpful if we want to build a classifier and keep only "good data" (*i.e.*, which discriminates between the classes we aim to infer) or if we want to understand the minimal resources (sensors and/or tasks) needed when performing data collection in a third-party, more resource-constrained, clinic.

We envisioned a visual analytics design that supports Shneiderman's mantra: Overview first, zoom and filter, then details-on-demand. Nevertheless, to generate interactive visualizations that implement these, we need to perform a series of transformations on the raw data. These are explained in detail next (see also Fig. 7.1 for an overview of the tool's pipeline).

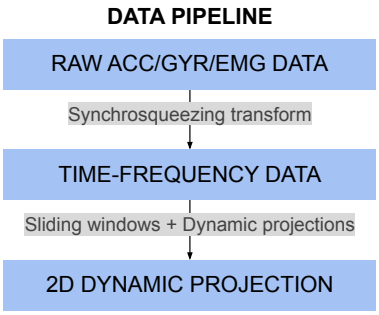


Figure 7.1: Data transformation pipeline.

7.4.1 *Raw data visual inspection*

To illustrate all the data transformations, we begin by inspecting the raw data given by an accelerometer sensor placed on the subject's right hand. The task we will investigate is described as "Arms stretched forward, wrists straight, palms down, and suppress involuntary movements" (Fig. 7.2).

Figure 7.2: Static position with arms and hands stretched out in front of body for 20-30 seconds. The subject tries to suppress any involuntary movements.

Fig. 7.3 shows the accelerometer data collected during the experiments for three subjects. For each subject, we see three colored curves corresponding to the X (red), Y (green), and Z (blue) axes measurements of a sensor placed on the right opisthenar (back of the right hand). The recording rate is 148 samples per second. The curves are offset from each other because the sensor reads the subject's movement acceleration as well as the Earth's gravity. An accelerometer at rest on the surface of the Earth will measure an upwards acceleration due to the gravity of $g \approx 9.81 m/s^2$.

The same sensor unit also records gyroscopic and EMG data. We decided to ignore EMG data due to the additional complexity introduced in the non-standard preprocessing steps that (usually) involve noise rejection/filtering, whitening, gain scaling, demodulation, smoothing, and relinearization. For the discussion next, we will only focus on accelerometer data.

The three subplots in Fig. 7.3 show data acceleration corresponding to three subjects:

- Patient *a* (top panel) is the *healthy* control: The curves are very close to straight lines. The small oscillations indicate very low-magnitude movement since it is impossible, even for a healthy person, to hold this position perfectly still.
- Patient *b* (middle panel) measurements show a very rhythmic and regular *tremor* of average magnitude. This patient was actually diagnosed with essential tremor, the most common trembling disorder, often confused with Parkinson's disease.
- Patient *c* (bottom panel) shows curves with high amplitude random non-rhythmic movements. This patient was actually diagnosed with *chorea*.

7.4.2 *Time-frequency data analysis*

The simple graphs in Fig. 7.3 are handy for reasoning about the disorders and understanding their behavior over time. However, they do not tell the whole story. We can get a different perspective on the data that reveals relevant information hidden in the raw signal by decomposing it into the frequencies that form it. This can be done in two ways, *i.e.*, using a frequency-domain representation or alternatively a time-frequency representation. The former assumes that the signal is stationary. Given the nature of our experiments, we know that our data does not fall in this category. Hence, we do not consider frequency-domain representations such as the Fourier Transform. In contrast, a time-frequency representation allows us to reason about how the frequency-domain (the spectrum) of a signal changes *over time*. We explore this representation next.

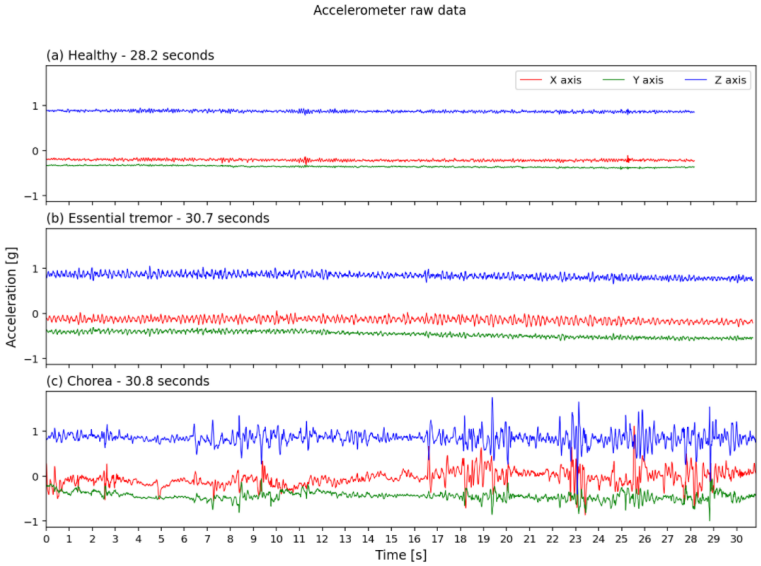


Figure 7.3: Accelerometer data for three subjects classified by experts as (a) healthy, (b) diagnosed with essential tremor, and (c) diagnosed with chorea. The data corresponds to a task where the subject must hold their hands as still as possible in the position suggested in Fig. 7.2. Each subplot corresponds to the accelerometer data collected during the experiment and is broken down into three orthogonal components (X, Y, and Z-axis).

The technique we use for time-frequency analysis is called Synchrosqueezing Wavelet Transform (SWT) (Mihalec et al., 2016). It is an improvement over the original Wavelet transform (Daubechies, 1990) which provides a sparser, sharper, noise-robust, and partly denoised representation of the time-frequency information. Figure 7.4, also called a spectrogram, shows a visual representation of the SWT for the data shown earlier in Fig. 7.3. The horizontal axis indicates, again, time. For every time moment, the vertical axis plots a frequency spectrum for that moment, where the magnitude (presence) of a certain frequency is color-coded (black is low, white is high, magnitude).

This time-frequency representation can be of great relevance for the diagnosis of motion disorders. For example, studies suggest that 95% of essential tremor cases exhibited frequencies in the 5–8Hz range. If we examine Fig. 7.4(b), we can see that, through the whole experiment, there is a defined presence of frequencies in this precise range, showing a common and insuppressible tendency to the involuntary movement. We do not see the same "horizontal line" in that frequency range for patients *a* and *c*. Patient *a* can hold his/her hand in a much more stable position, which translates into a darker spectrogram (less energy).

In contrast, patient *c* performs high amplitude random non-rhythmic movements characteristic of chorea. The light colors in the spectrogram represent the high amplitude, and the "random non-rhythmic" aspect can be read as having no constant lines in the time-frequency representation, as seen on patient *b*. Having both representations (Figs. 7.3 and 7.4) side-by-side helps us understand the phenomena as a whole.

For our purposes, the time-frequency representation is essential as it facilitates comparison between signals. Methods for computing similarity of signals in the time-domain (raw signals) exist, *e.g.*, RMSE, cross-correlation, and Dynamic Time Warping (Gupta et al., 1996)). However, these can be significantly affected by the phase shift and minor differences in frequency. By making comparisons in the time-frequency domain, we get a better, more reliable representation that is better suited for the next steps of our pipeline.

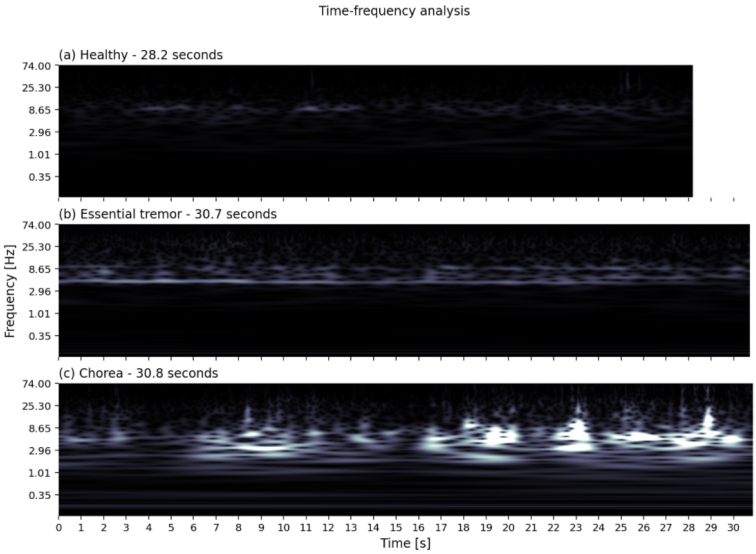


Figure 7.4: Time-frequency representation of the acceleration of the Z axis (blue) in Fig. 7.3. This is the vertical axis which points up/down for the hands shown in Fig. 7.2. Light colors represent high energy in the spectral distribution, *i.e.*, there is a large amplitude component in the subject's movement associated to a particular frequency or frequency distribution.

7.4.3 *Data normalization*

The acceleration time plots and spectrograms discussed so far are quite effective in studying the data of a *single* subject. However, as outlined earlier in the chapter, our goal is to compare *multiple* subjects in order

to find out which data aspects make them similar or different. For this, we need a way to 'normalize' the collected data collected from each measurement (experiment) so that it becomes comparable across multiple measurements.

As seen in Fig. 7.4, time-frequency information can vary slightly from recording to recording. These recordings can have different lengths and different frequency ranges. Both of these are due to the difference in experiment duration. Experiments that last more have longer spectrograms and a higher frequency range since we can detect more frequencies in the lower range. The sampling rate of the sensor sets the upper range of the frequency spectrum. The accelerometers we used can collect 148 readings per second, so the maximum frequency we can detect is 74 Hz. We also limit our frequency range to 100 logarithmic bins in the range of $0 - 74$ Hz.

To get our data in a uniform and easily comparable format, we use the Sliding Window method as illustrated in Fig. 7.5. In this example, we set a stride t_s of 1 second and a time-window width t_w of 5 seconds. Each such time window will thus aggregate the data falling within it into a single (high-dimensional) measurement. Increasing t_w introduces more filtering, which can be desirable when the data is highly noisy (or we are interested mainly in lower frequencies). Increasing t_s decreases the total number of sample points (along the time axis) that we reduce our data to. We can change the parameters t_s and t_w on the fly to generate visualizations of different "resolutions", as discussed next.

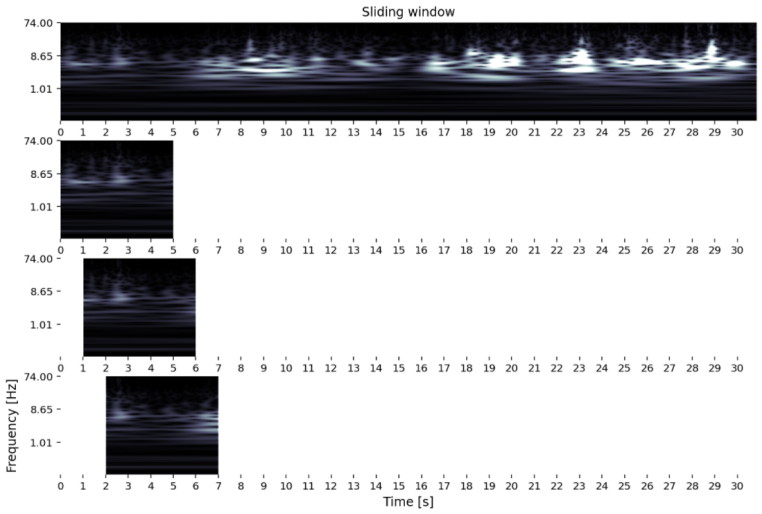


Figure 7.5: Before we project our data, we subdivide each spectrogram using the Sliding Window method. In this example, we use a window width of $t_w = 5$ seconds and a stride of $t_s = 1$ second to partition the data from Fig. 7.4c.

7.4.4 *Visualizing the data with dynamic projections*

Since our data is multidimensional and temporal, we use dynamic projections to obtain further insights into our dataset. For this tool, we implemented three of the dynamic projection methods introduced in the previous two chapters. These are G-PCA, G-tSNE (Chapter 5), and PCD-tSNE (Chapter 6). As discussed earlier in the thesis, G-PCA and PCD-tSNE are methods that balance visual quality and temporal stability, but they each have pros and cons. PCD-tSNE has better neighborhood and distance preservation than G-PCA and comparable stability. Its main drawback for this particular application is that a few parameters need to be adjusted, which adds an undesired level of uncertainty when interactively exploring the data. This same trait outside of an interactive setting can be advantageous, as it allows us to create projections that borrow characteristics from both PCA and tSNE-based methods, as previously discussed in Section 6.5. G-tSNE is a relatively unstable technique, as benchmarked in Chapter 5. However, it still provides interesting (non-temporal) insights into the high-dimensional structure of the data.

Fig. 7.6 shows the data from patients *a*, *b*, and *c* projected using the three DR methods mention above. In each projection, we see three curves (trails). Each curve describes the behavior over time of one patient – simply put, the patient can be seen as 'moving along the curve' in projection space. As the spectral signatures of the different patients are different from each other, we see that the trails do not overlap. We also observe how PCD-tSNE creates a projection that borrows traits from both G-PCA and G-tSNE. The general position of all clusters and shape of the (c) trail resemble that of G-PCA, while the focus on inter-cluster neighborhood preservation for trails (a) and (b) (materialized as expanded clusters) are traits derived from the t-SNE influence.

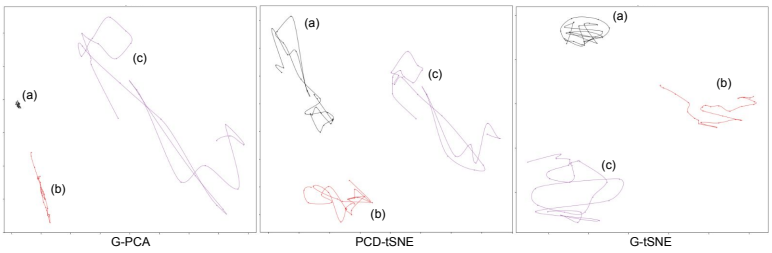


Figure 7.6: The last step of the pipeline is to project (using dynamic methods) the data that has been subdivided by the Sliding Window method. Our tool supports 3 dynamic projection methods: G-PCA, PCD-tSNE, and G-tSNE. We visualize the dynamic projections as trails, where we connect the consecutive spectrum "windows" via Akima spline (Akima, 1970). The data corresponds to the patients presented in Figs. 7.3 and 7.4.

7.5 DATA EXPLORATION

This section presents an example of the type of data exploration that our visual analytics tool supports.

There are many ways to conduct data exploration, depending on the goal of the analysis. Some relevant examples from a clinical standpoint are

- understand the behavior of different patients or patient groups for a given task;
- given new patient measurements, find patients with similar movement characteristics to assist diagnosis;
- explore the movements that led to a specific diagnosis;
- study the erratic/circumstantial nature of the appearance of certain symptoms;
- understand how different tasks induce different behaviors on specific patients or patient groups.

In the following, we will describe an example analysis that will focus on the first goal. We want to understand the behavior of essential tremor patients *vs* healthy controls in the context of the task described as "arms stretched forward, wrists straight, palms down, and suppress involuntary movements" (Fig. 7.2). In total, we have 24 healthy subjects and 11 tremor patients. For the next figures, we will select to use the reading from the Z-axis of the accelerometer sensor placed on the subject's right hand.

Once we select the data subset of interest, we need to choose how to project the data. Fig. 7.7 displays two G-PCA projections. The only difference is the size of the sliding window t_w (Fig. 7.5), which translates into the resolution and smoothness of the projection. The left projection has a sliding window size of $t_w = 1$ second and a stride of $t_s = 1$ second, meaning that there is no overlap between the consecutive windows, leading to more jagged trails. The right projection has a window size of $t_w = 5$ seconds instead, so consecutive windows share part of the same data, leading to a longer representation of temporal phenomena and smoother trails. The choice of window size and window stride also depends on the amount of data we are projecting, *i.e.*, number of patients and length of experiments. We will use an empirically found window size of $t_w = 5$ seconds and a stride of $t_s = 1$ second for the following figures.

With the projected data in front of us, we can start making some observations. We see that most healthy patients (black trails) cluster together in a tight formation in the "center" of the projection, while the tremor patients' trails tend to be located to the right of this cluster and

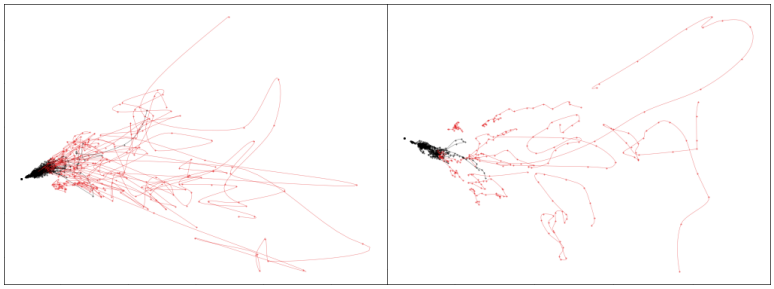


Figure 7.7: G-PCA projection of 24 healthy (black) and 11 tremor (red) patients with window width and stride of [1s, 1s] on the left and [5s, 1s] on the right.

have a broader spread. The angle and distance from the central cluster must portray some traits in the data, but we cannot confirm anything just yet.

To better understand the overall structure of the projection, we start by looking at a couple of trails drawn in its periphery. We start selecting patient 91 (Fig. 7.8-left). Our tool also displays the video recording of the experiment (not shown here due to privacy concerns).

Patient 91 presents a unique behavior. Looking at the raw signal and spectrogram, we can tell that in the initial 2-3 seconds, there is a presence of medium amplitude tremors, followed by high amplitude tremors until second 13, and then a significant reduction in the intensity until the end of the experiment. We also see that the movement is decomposed into a "sharp" spectrum, mainly focused on around 5 Hz. One hypothesis is that the patient can suppress involuntary movements after considerable effort, which is only possible after a few seconds into the experiment. This dynamic translates in a particular projection trail (top-left): The trail starts a certain distance away from the center of the projection, characteristic of "unhealthy" behavior, and then "shoots" away to the top left as tremor intensity increases, only to finally move closer to the center of the projection, close to the healthy cluster as the amplitude is reduced. This hints that the distance from the "center" of the projection is related to the energy in the signal, that is, how large the movements are.

The three plots on the right portray the data collected from patient 96 (Fig. 7.8-right). This patient was also diagnosed with important tremors, but its manifestation shows essential differences compared to patient 91. Patient 96 shows a more constant tremor. The amplitude is constantly high, showing that the patient cannot suppress the movement. Another dissimilarity comes in the signature of the tremor: Patient 91 has a "sharp" histogram, meaning that his tremor has very sinusoidal tendencies, while the same is not valid for patient 96, given the presence of high energy high frequencies on the spectrum. Upon detailed

inspection of the video recording, this can be related to the fact that patient 96's tremor has a more lateral tendency (side-to-side instead of up-and-down). We also notice extra movement at the beginning of the recording as the patient gets into position, which can be identified in the projection as abnormal movements that make the trails start at the right-bottom at a considerable distance from where the rest of the trail is located.

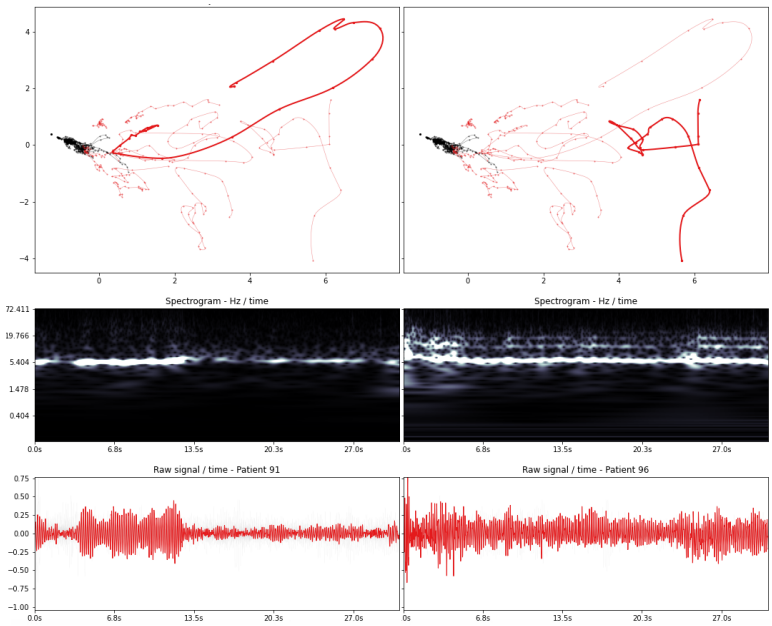


Figure 7.8: Our tool allows selection and inspection of patients. Other than the plots shown in the figure, the tool also displays a video recording of the experiment.

The projection also allows us to investigate the border between the two classes, potentially revealing subjects of interest with complex diagnoses. Fig. 7.9 shows two subjects whose trails are in the border and overlap each other. However, patient 39 (left) is classified as healthy, while patient 42 (right) was diagnosed with essential tremor. Comparing their signals, we see that subject 39's signal has a larger amplitude and a sharper spectrum, which to the untrained eye, it could mean that he/she suffers from tremors. However, the subject's tremors have a very high-frequency main component (over 10 Hz), which puts him/her outside the normal range for essential tremors (5–8 Hz) – maybe the subject just drank too much coffee in the morning. His/her tremor also does not seem to be a very significant impact on other tasks. Patient 42, however, does not display obvious tremor symptoms in this particular static task, but given its classification, we must explore the reasons for the diag-

nosis. Further exploration tells us that this particular patient has one side of the body more affected than the other. The condition appears to be more noticeable when performing active tasks (instead of holding static positions), as witnessed by the so-called Archimedes Spiral tests[2] (Fig. 7.10). Patient 42 shows tremor signals for both hands, but the tremor is much more apparent on the left.

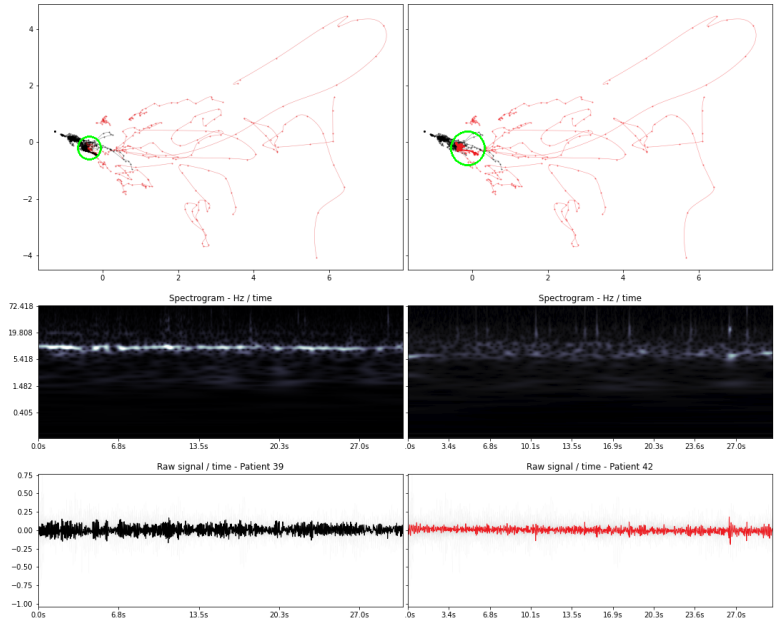


Figure 7.9: The trail representing the patient on the right is very close to the healthy group. If we look at only his right hand during the recording of this specific task, it is hard to tell that he/she suffers from tremors, and it raises the question as to why was he/she diagnosed with tremors. Does it have a postural/unilateral aspect that is not captured in this task?

Finally, we can notice some healthy trails reaching out into "tremor territory", revealing interesting subjects for investigation. One trail that attracts attention in the projection is the one of patient 3 (Fig. 7.11-left). While most of the trail is located near other healthy patients, a part of the trail extends to the right. Looking at the spectrogram and raw signal, we can see that the patient trembles for a short period. However, the video (again, not shown here for privacy concerns) shows that this happens as the patient talks to the researchers and slightly moves his/her legs, which does not indicate an incorrect diagnosis. Such a trail could also mean that a true essential tremor patient lost control of the suppres-

2 In this drawing task, the subject is shown an Archimedean spiral, and asked to reproduce it as faithfully as possible (Bain et al., 1993).

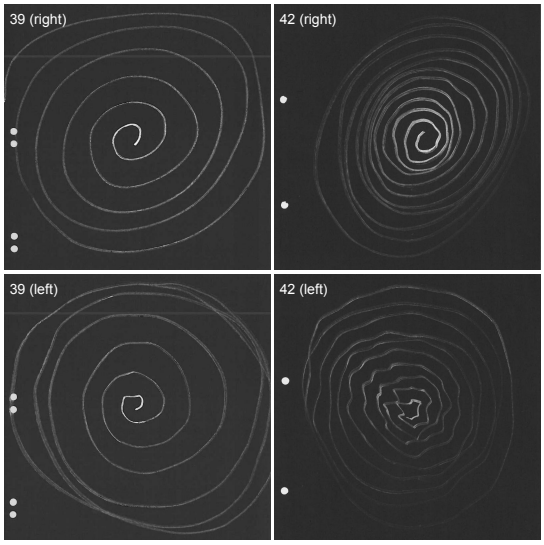


Figure 7.10: Archimedes spirals for both hands of patients 39 and 42.

sion of the involuntary movement. The moment this happens is clearly pointed out in the projection and could have been easily not noticed in a clinical setting. We see a similar trend for patient 37 (Fig. 7.11-right), but in this case, the movement is due to a late stop of the recording – the sensors and camera keep going as the subject is told to return to a rest position. These recordings also help us confirm an earlier suspicion about the meaning of the northeast and southeast portions of the projection space. These are related to the "sharpness" of the spectrum: signals where the frequency band of 5–8 Hz tends to form the majority of the spectrum energy tend to go north-east, these "well-behaved/well-defined" tremors. At the same time, the direction towards the southeast of the projection is representative of more uniform spectral distributions, related to more chaotic movements.

All the previous analysis was done atop of a single G-PCA projection. However, the other projection techniques implemented into the tool can offer different perspectives on the data, which could be useful in other tasks. Fig. 7.12 shows the results of six projection techniques for the same data. These projections show a PCD-tSNE characteristic previously discussed (Section 6.5.4), namely its ability to mix characteristics of PCA (focus on distance preservation) and of tSNE (focus on neighborhood preservation) based on the setting of its λ parameter.

Our analysis so far focused on comparing control subjects to tremor patients. Another important clinical question concerns comparing different disorders and patient groups to comprehend better disorder manifestation and support challenging phenotypic classification. As previously pointed out, cross-analysis is an understudied topic supported

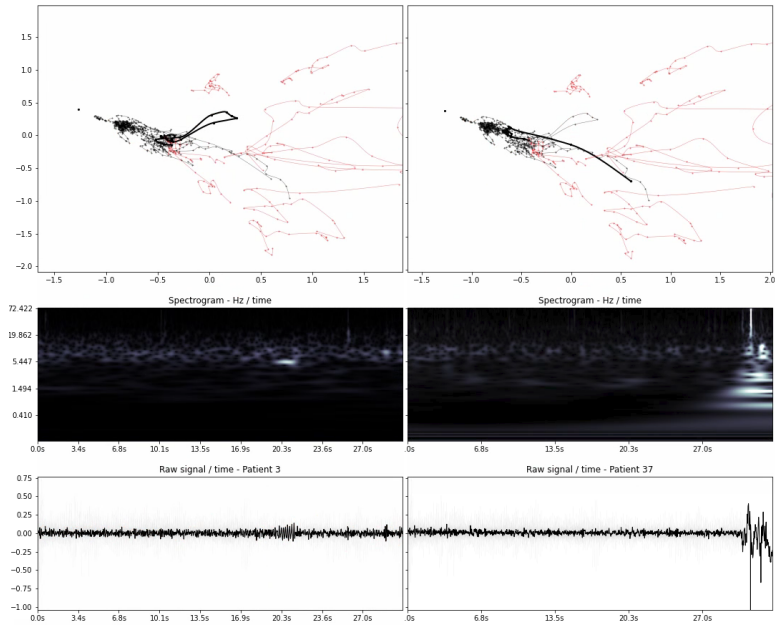


Figure 7.11: Temporary tremors on the left, and premature end of experiment on the right.

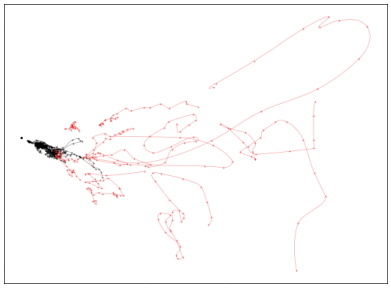
by our tool. Fig. 7.13 shows patients classified with Myoclonus and Essential Tremors on the same projection, referring to a relevant clinical question, given that Myoclonus can manifest itself in the form of periodic jerks that closely resemble tremors. Being able to perform the phenotypic classification correctly is critical for effective treatment of the disorders. In the projections of this data, we can see some separation in the two classes. This is valuable evidence for the value of our projection-based approach, supporting our claim that it can lead to further developments in classification development.

7.6 DISCUSSION

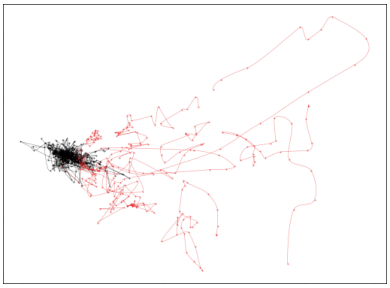
Given the novelty of our approach and the initial positive results obtained, several points can be discussed to improve our visual analytics tool to make it more effective for tackling phenotypic classification and exploration of EMG and accelerometer data. Below we discuss several such points and indicate directions for future work and further exploration.

- *What sort of insights can we get about undiagnosed patients?*

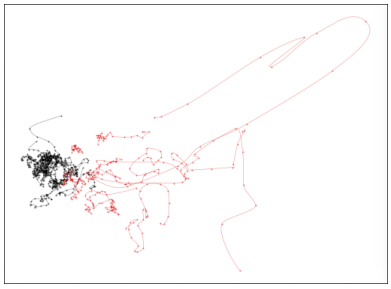
One possible use of our tool, which has not been properly investigated yet, is its capability as a visual *classification* tool. In practice, this could



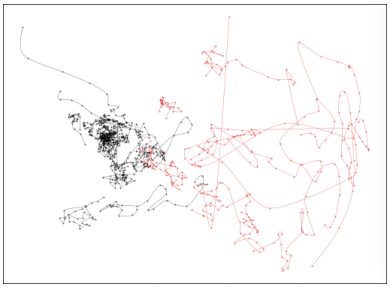
(a) G-PCA



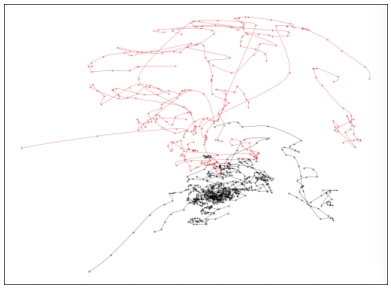
(b) PCD-tSNE $\lambda = .1$



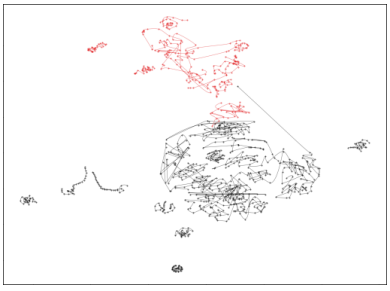
(c) PCD-tSNE $\lambda = .001$



(d) PCD-tSNE $\lambda = .0001$

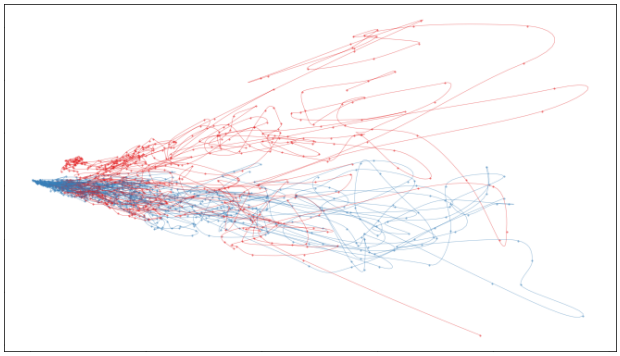


(e) PCD-tSNE $\lambda = .000001$

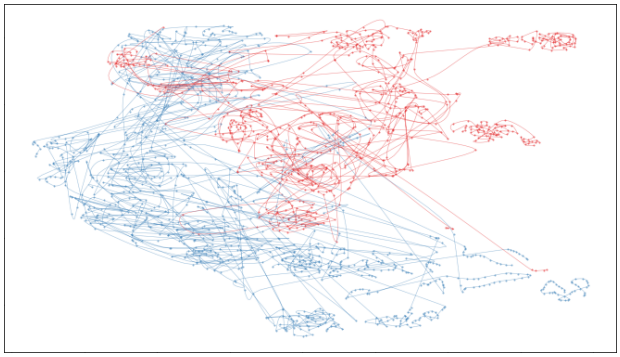


(f) G-tSNE

Figure 7.12: Different projection of the same data showing how PCD-tSNE is able to create a hybrid focus on distance preservation or neighborhood preservation depending on the λ parameter setting.



(a) G-PCA



(b) G-tSNE

Figure 7.13: G-PCA and G-tSNE projections of 16 Myoclonus patients (blue) and 11 essential tremor (red) patients for the same task and sensor placement with window size of 1s and stride of 0.5s. Some separation can be seen, which is a good indicative that the methods we developed can be used to support the classification of patients groups, which is one of the ultimate goals of the NEMO project, but falls outside the scope of this thesis.

be done by adding a new undiagnosed study subject to a projection such as the one in Fig. 7.13. Given the data transformations performed to construct the lower dimensional (embedding) space and/or the similarity of the subject's data compared to the other diagnosed recordings, by comparing the shape and position of the new trail to the ones already present in the projection, we can find evidence that may lead to a diagnosis (classification). Given the large number of experiments each patient performs, by using multiple projections and comparison groups, we could also draw conclusions about the circumstantial and individual aspects of the excessive and involuntary movements.

- *Can we identify data features that explain why patients or patient groups differ from each other in the projections?*

We addressed this goal by designing a tool that explores the data "both ways". In the "forward direction", we can take the raw data and, via many transformations and augmentation steps, create a high-level complex representation using projection trails that allow us to reason about the underlying temporal and high-dimensional structure of the dataset. The "reverse direction" implies taking this complex visual representation, and via interaction and group inspection, explore the features on the original data that cause the high-level representation to present specific visual features. The analysis performed on Figs. 7.7 and 7.8 is a prime example of such feature identification, as the exploration we performed quickly gave us insight into the intensity (given the distance of trails to the control group) and regularity (based on the orientation of trails) of tremors.

- *Can we use the tool to identify which task and sensor combinations are relevant to a given diagnosis?*

Due to the size and goals of the NEMO project, we were able to collect large amounts of data from patients, asking them to perform many motion tasks, while connected to a range of state-of-the-art sensors. This is, of course, taxing on the subjects. With that in mind, and knowing that most clinical settings are not as well equipped or prepared to run such tests, one of the clinical goals of this project is to define what are (if any) the *minimal* resources and methods needed to perform a confident data-driven diagnosis. This means finding combinations of sensors, tasks, comparison groups, and (pre)processing steps, that are the most suitable in support of clinical tasks.

By creating an interactive tool in which the user can adjust all these settings and parameters on the fly, while receiving visual feedback in the form of projection trails and secondary views of the data, we give the user new ways of reasoning and interacting with the data. We also believe that the separation and visual features seen in the projections are meaningful and relate to clinical features. For example, if a combination of sensors, tasks, and user groups seem to produce good class

separation in the projection, we can expect a black-box classifier to perform well in this data (Rauber et al., 2017a) and we expect experts to also agree with the overall structure of the data.

- *Can we use the tool to make data quality judgements?*

Indeed, as shown in Fig. 7.11, we performed an analysis on subjects that were visually found out to deviate from the healthy group, in which we found that the duration of the experiment for subject 37 deviated from the duration of the recording. In the same figure, we see a small black dot to the left of the control group that indicates the recording of a patient for which this particular sensor was not correctly attached.

- *How should we continue development of this tool?*

There are many promising directions for future development that address current limitations in our tool: For example, all our current analyses are now done based on *one* selected task and *one* sensor at a time. We believe that it would be advantageous to perform a unified analysis on multiple sensors and tasks. This could lead to a better understanding of postural/circumstantial (depending on the task or patient focus) and individual (unilateral or only certain limbs affected) appearance of certain involuntary movements.

Additionally, in its current state, our tool performs well only for static tasks, i.e., tasks in which patients are expected to hold a position for a specific amount of time. However, many dynamic tasks are of great relevance for specific diagnoses, but due to the complexity of the signal, we are unable to extract relevant information for motion disorder analysis. Correctly handling this data would imply the development of a series of additional preprocessing steps that fall outside this thesis's scope.

Given the goals of the NEMO project, it is important to face the problems of visual analytics and classification as one. This chapter focuses on the former, but there are works in the literature that present combined approaches (Graving and Couzin, 2020; Rauber et al., 2017a) which could be of benefit to the clinical tasks we want to improve.

Lastly, we focus only on EMG, accelerometer, and 2D video data. It could be beneficial to add support to exploring the extra data modalities extracted during the experiments.

7.7 CONCLUSIONS

We presented a real-world application of our new dynamic projection methods in the context of hyperkinetic movement disorder analysis. We show evidence of the usefulness of these methods in support of motion data analysis.

We introduced the problem, described how we transform the data collected during clinical experiments, proposed a visual analytics tool

designed to support the exploration of this complex dataset, and showed examples of data explorations that led to valuable clinical insights.

This is a preliminary investigation, and many points are still open, e.g., the actual construction of an automatic classifier based on such data, questions regarding the clinical usage of dynamic projections, the design of sophisticated UIs for medical professionals, among other directions of future work. Nevertheless, our work showed that projections have the potential to be useful in exploring temporal multidimensional data coming from a complex real-world problem.

Acknowledgments: I would like to acknowledge the University Medical Center Groningen (UMCG) for providing access to the NEMO data and the researchers M.A.J. de Koning-Tijssen, J.R. Dalenberg, A.M.M. van der Stouwe, I. Tuitert, A.C. Telea, and J.L.D. Comba for contributing to the writing of this chapter.

8

CONCLUSION

In this thesis, we considered the visualization of two types of dynamic (time-dependent) data present in information visualization – weighted hierarchies and high-dimensional datasets. Both data types are ubiquitous in many applications in machine learning, statistics, and data science; and, as identified at the beginning of the thesis, while many techniques have been proposed for the visualization of static (time-independent) forms of both data types, the investigation of techniques that handle the dynamic variants has only been touched in information visualization.

Throughout our work, we found interesting and insightful parallels between the two types of datasets, the challenges they pose to visualization and the visualization evaluation, and also the solutions that we designed to handle both.

Chapter 2 kickstarted our work by considering treemap algorithms for the visualization of a particular type of data, namely hierarchies mined from evolving software projects. Already in this limited context, we found that quality of a treemapping algorithm contains two components, namely *visual quality*, that captures how well the cells of the treemap are spread over the drawing space to reflect the data values and also generate easily readable patterns; and *stability*, that measures how well the changes in the depicted treemaps follow the changes in the underlying hierarchies. We also found that the two quality aspects are, roughly speaking, in competition with each other: Algorithms that obtain a high visual quality do this by neglecting stability; and very stable algorithms yield a poor visual quality.

For both hierarchies and high-dimensional projections, we found well-established metrics for gauging visual quality in the literature, *i.e.*, when time is not taken into consideration and only the quality of individual layouts is measured. For treemaps, the quality of the layout is well quantified by the aspect ratio of the contained cells, considering that cells closer to squares form a more readable visualization; for projections, visual quality is measured by how well the distances and neighborhoods from the high-dimensional space are preserved by the low-dimensional embedding.

Regarding stability, however, there were no effective methods of quantifying the relationship between *data change* and *visual change*. As such, and recognizing that stability is an as important desirable property for dynamic visualization as their (static) visual quality, we proposed our own stability metrics. Concerning dynamic treemaps, an algorithm is stable if small changes in the input data result in small changes in

the layout, that is, data change and layout change correlate positively. Previously proposed stability metrics measured only the layout change and concluded that small layout changes are a sign of a stable algorithm. However, to properly measure stability, we also need to capture the data change and then correlate data and layout change, an endeavor which we approached in Chapters 2 and 3. This exact same principle applies to dynamic projections. However, in this case, we correlate high-dimensional data change to low-dimensional scatterplot layout change (Chapter 5).

As already mentioned, we found out that visual quality and stability are conflicting criteria, both for treemaps and projections. In order to improve stability, both treemapping and dimensionality reduction methods have to sacrifice visual quality, and conversely. Recognizing this challenge, we next aimed to create methods that improve this balance – that is, yield overall higher stability and visual quality than existing methods in each class. For dynamic hierarchies, we proposed to this end Greedy Insertion Treemaps (Chapter 4). For multidimensional projections, we proposed the LD-tSNE and PCD-tSNE methods (Chapter 6). Greedy Insertion Treemap (or GIT) aims to preserve treemap-cell neighborhoods over time by constructing an initial so-called Layout Tree (LT), a data structure which is incrementally updated as the input tree data changes, so as to minimize undesired treemap-layout changes. Our state-aware GIT method is simple to implement, generic (handles any types of dynamic hierarchies), and fast (compared to the other state-of-the-art methods). For the dynamic projection challenge, both our newly proposed methods leverage the neighborhood-preservation ability of t-SNE for dynamic time-dependent data. LD-tSNE uses guidance in the form of landmarks, and PCD-tSNE uses information given by the Principal Components of the full temporal dataset. Our results show that PCD-tSNE scores a good balance between stability, neighborhood preservation, and distance preservation, making it one of the best suited general methods for dynamic projections, while LD-tSNE allows creating stable and customizable projections via landmarks selection and steering.

Another common aspect concerning both dynamic hierarchies and dynamic high-dimensional datasets is the difficulty of *evaluation*. This comprises multiple aspects. Besides the availability of suitable quality metrics – which we solved as described above – there is also the difficulty of finding good collections of datasets on which to evaluate existing visualization methods. Such so-called benchmarks were introduced – only very recently – for static projections. However, no comprehensive benchmarks existed, at the time of writing this thesis, for static treemapping, let alone for dynamic treemapping and dynamic projections. We created and evaluated several such benchmarks, starting with one for dynamic hierarchies obtained from software evolution (Chapter 2), which we extended next to a far more general benchmark for dynamic hierarchies mined from a wide spectrum of application do-

mains (Chapter 3), and ending with a benchmark for dynamic projections (Chapter 5). Creating these benchmarks posed both conceptual problems, in terms of how to describe the huge variability of dynamic datasets along a set of independent traits, and next how to sample these traits; but also practical problems, in terms of how to find real-world datasets that sample the universe of these dynamic datasets, and providing actual reference implementations for the tens of algorithms for treemapping and projection that we need to evaluate. While our proposed benchmarks are, definitely, not fully covering the space of possibilities, they are the first in the dynamic treemapping and projection arenas. We made them fully open source (datasets, algorithms, visualization techniques, quality metrics, and obtained results). We argue that these are important resources for the visualization community which can, now, easily compare new and existing algorithms with new and existing datasets for both practical and research-oriented goals.

Lastly, we presented a real-world application our new dynamic projection methods in the context of hyperkinetic movement disorder analysis. These disorders manifest as abnormal involuntary movements that highly affect the quality of life of the people who suffer from them, and computer supported diagnosis is desired given the complexity of their manifestation. In Chapter 7, we described how we transform the data collected during clinical experiments, we proposed a visual analytics tool designed to support the exploration of the this complex dataset, and we showed an example of data exploration that leads to valuable clinical insights. This is preliminary investigation, and many points are still open e.g., the actual construction of an automatic classifier based on such data, questions regarding the clinical usage of dynamic projections, the design of a sophisticated UIs for medical professionals, among other directions of future work. Nevertheless, our work showed that projections do have the potential to be useful in the exploration of temporal multidimensional data coming from a real-world problem.

8.1 FUTURE WORK

There are several possible directions for future work:
Streaming data: For our algorithm designs, we assumed a finite temporal aspect to the time series we are handling, and we allow our algorithms to "look into the future" and adapt accordingly to the changes in the data that are yet to come. When dealing with streaming data, we don't have this ability, we can't "look into the future", and the algorithm must try to adapt to any unpredictable changes in the data, making the design of this class of methods even more challenging. In addition, studying this class of (underserved) algorithms implies the design of new quality metrics and the collection of new suitable datasets. This applies both to streaming treemaps and streaming projections.

Deep learning dynamic projections: Recently, we have seen the use of deep neural networks to produce static projections with a significant computational speed-up while maintaining high-quality metrics and out-of-sample capability (Espadoto et al., 2020b). We believe a similar approach could be investigated for dynamic projections, granting similar benefits to the temporal counterpart.
Extending benchmarks: We can extend our benchmarks with new methods, better ways to choose hyperparameters, new datasets, and new metrics. With a larger number of datasets, we can perform robust tests on the impact of dataset traits on the quality of our projections and treemaps. We can also integrate streaming data techniques, streaming datasets, and dedicated task-based tests.
Improvements to the NEMO data exploration tool: As stated in Chapter 7, there are many promising direction for future development that address current limitations in our tool: the current analyses only support *one* selected task and *one* sensor at a time. It would certainly be advantageous to extend our methods to multiple sensors and tasks. This would lead to better understanding of the postural, circumstantial, and individual aspects of certain involuntary movement occurance. Additionally, we focus only on EMG, acceleremoter, and 2D video data. It would certainly be beneficial to the support of phenotypical classification to add support to the extra data modalities extracted during the experiments.

A

APPENDIX: GUIDED STABLE DYNAMIC PROJECTIONS

A.1 PCD-TSNE PARAMETERS

Table 7 presents the PCD-tSNE parameters used for each dataset. This table complements Sec. 3.2 of the main text.

dataset	λ	PC scaling
cartolastd	10^{-2}	10^{0}
cifar10cnn	10^{-5}	10^{0}
esc50	10^{-2}	10^{-1}
fashion	10^{-4}	10^{-1}
gaussians	10^{-3}	10^{1}
nnset	10^{-3}	10^{0}
qtables	10^{-3}	10^{-1}
quickdraw	10^{-3}	10^{0}
sorts	10^{-1}	10^{0}
walk	10^{-4}	10^{0}

Table 7: The λ parameter modulates the amount of global influence applied to points in $P(\mathbf{D}^t)$; the *PC scaling* term scales W to increase/decrease the area of global influence, i.e., it scales the principal components of $\mathbf{D}$.

A.2 LD-TSNE PARAMETERS

Table 8 presents the LD-tSNE parameters used for each dataset. This table complements Sec. 3.1 and Sec. 5.5 of the main text.

A.3 METRIC RESULTS

Table 9 shows unaggregated metric results. Each of the 10 subtables correspond to a dataset, columns correspond to the different quality metrics, and the rows represent the different methods. Methods are ordered according to their strategy: Per-timeframe, Global, Continuous, and Guided. The columns correspond, respectively, to distance preservation metrics ($S_{Pearson}, S_{Spearman}, S_{Kendall}, S_{Stress}$), neighborhood preservation metrics ($S_{NH}, S_{NP}, S_{Trust}, S_{Cont}$), and temporal stability metrics ()$T_{Pearson}, T_{Spearman}, T_{Kendall}, T_{Stress}$). The colormap is normalized independently for each metric and each dataset.

These tables compliment Sections 5.2 and 5.3 of the main text.

dataset	λ	β	α	$\mathbf{l}^q$ projection (# landmarks)
cartolastd	.2	2	4	PCA(N)
cifar10cnn	.5	4	1	tSNE(N)
esc50	.3	5	1	PCA(N)
fashion	.1	4	2	tSNE(N)
gaussians	-	-	-	PCA(N)
nnset	.02	8	1	PCA(N)
qtables	-	-	-	PCA(N)
quickdraw	.1	2	1	tSNE(N)
sorts	.25	10	2	PCA(NT)
walk	-	-	-	PCA(N)

Table 8: The λ, α, and β parameters control the amount of influence landmarks have on the the points being projected. In simple terms, α controls the tightness of clusters in $P(\mathbf{D}^t)$, β scales the strength of the "pull" of landmarks $\mathbf{L}$ on points in $P(\mathbf{D}^t)$, and λ balances the two factors. Values marked "-" were obtained using the interactive mode that was implemented and gave the user real-time control over parameters during the optimization. N is the number of points in $\mathbf{D}^t$ and T is the total number of timesteps in $\mathbf{D}$.

	$S_{Pearson}$	$S_{Spearman}$	$S_{Kendall}$	S_{Stress}	S_{NH}	S_{NP}	S_{Trust}	S_{Cont}	$T_{Pearson}$	$T_{Spearman}$	$T_{Kendall}$	T_{Stress}	
TF-PCA	**0.931**	0.928	0.790	**0.137**	0.505	0.480	0.937	0.876	0.761	0.570	0.450	0.477	
TF-tSNE	0.756	0.800	0.615	0.487	**0.597**	0.592	0.947	0.913	0.061	0.075	0.055	1.876	
TF-UMAP	0.634	0.693	0.520	0.731	0.576	0.556	0.908	0.893	-0.00	-0.04	-0.03	2.003	
G-AE	0.898	0.936	0.799	0.203	0.495	0.505	0.932	0.884	0.758	0.985	0.908	0.482	
G-VAE	0.910	**0.949**	**0.822**	0.178	0.568	**0.618**	0.950	**0.934**	**0.864**	**0.987**	**0.917**	**0.270**	
G-PCA	0.929	0.926	0.787	0.140	0.519	0.474	0.935	0.874	0.778	0.987	0.916	0.442	
G-tSNE	0.685	0.733	0.547	0.628	0.550	0.455	0.847	0.806	0.514	0.788	0.655	0.970	
G-UMAP	0.599	0.635	0.459	0.801	0.290	0.156	0.561	0.576	0.366	0.368	0.282	1.267	
C-tSNE	0.602	0.678	0.495	0.794	0.527	0.503	0.849	0.846	0.307	0.175	0.132	1.384	
C-UMAP	0.665	0.711	0.533	0.668	0.576	0.555	0.923	0.895	0.044	-0.11	-0.08	1.911	
D-tSNE	0.768	0.822	0.638	0.462	0.538	0.558	0.932	0.905	0.141	-0.03	-0.02	1.716	
PCD-tSNE	0.929	0.924	0.785	0.141	0.521	0.478	0.936	0.875	0.772	0.704	0.565	0.454	
LD-tSNE	0.771	0.823	0.642	0.457	0.560	0.573	**0.957**	0.907	0.574	0.392	0.301	0.851	

cartolastd

TF-PCA	0.786	0.790	0.600	0.427	0.414	0.560	0.942	0.894	-0.11	-0.13	-0.08	2.229	
TF-tSNE	0.786	0.783	0.597	0.427	**0.479**	**0.776**	0.963	**0.972**	-0.13	-0.15	-0.10	2.272	
TF-UMAP	0.845	0.855	0.672	0.308	0.475	0.752	0.960	0.967	-0.04	-0.06	-0.04	2.096	
G-AE	0.772	0.792	0.604	0.454	0.455	0.621	0.931	0.922	0.597	0.750	0.559	0.804	
G-VAE	0.905	0.917	0.762	0.188	0.469	0.732	**0.969**	0.944	**0.816**	**0.889**	**0.720**	**0.367**	
G-PCA	0.777	0.790	0.602	0.445	0.405	0.532	0.935	0.881	0.695	0.654	0.480	0.608	
G-tSNE	0.886	0.884	0.713	0.226	0.455	0.603	0.903	0.851	0.521	0.641	0.460	0.957	
G-UMAP	0.906	**0.922**	**0.766**	0.186	0.468	0.694	0.953	0.942	0.682	0.710	0.517	0.634	
C-tSNE	0.888	0.916	0.752	0.223	0.455	0.623	0.897	0.887	0.378	0.367	0.251	1.243	
C-UMAP	0.831	0.853	0.669	0.336	0.474	0.747	0.956	0.967	0.346	0.189	0.127	1.307	
D-tSNE	0.842	0.845	0.662	0.315	0.476	0.721	0.952	0.950	0.317	0.283	0.191	1.365	
PCD-tSNE	**0.908**	0.913	0.753	**0.182**	0.459	0.647	0.914	0.889	0.483	0.497	0.348	1.032	
LD-tSNE	0.865	0.870	0.691	0.268	0.465	0.704	0.942	0.952	0.570	0.691	0.502	0.858	

cifar10cnn

TF-PCA	**0.993**	**0.990**	**0.927**	**0.012**	0.323	0.712	**0.989**	**0.974**	0.095	0.801	0.641	1.808	
TF-tSNE	0.920	0.935	0.784	0.159	0.372	0.694	0.964	0.969	-0.04	-0.03	-0.02	2.096	
TF-UMAP	0.926	0.935	0.786	0.146	0.360	0.676	0.967	0.968	-0.10	-0.05	-0.03	2.214	
G-AE	0.937	0.965	0.842	0.124	0.323	0.604	0.961	0.959	0.981	0.898	0.738	0.037	
G-VAE	0.977	0.968	0.859	0.045	0.210	0.458	0.939	0.896	0.985	0.866	0.697	0.029	
G-PCA	0.993	0.990	0.926	0.012	0.323	0.711	0.989	0.974	**0.992**	**0.917**	**0.778**	**0.014**	
G-tSNE	0.776	0.827	0.639	0.446	**0.393**	0.356	0.748	0.769	0.608	0.545	0.386	0.783	
G-UMAP	0.778	0.812	0.616	0.443	0.300	0.320	0.754	0.769	0.568	0.550	0.392	0.862	
C-tSNE	0.887	0.909	0.742	0.225	0.367	0.685	0.960	0.964	0.384	0.557	0.394	1.231	
C-UMAP	0.928	0.936	0.787	0.142	0.359	0.668	0.967	0.967	0.034	0.029	0.019	1.930	
D-tSNE	0.912	0.925	0.771	0.174	0.305	0.510	0.938	0.929	0.026	0.020	0.013	1.947	
PCD-tSNE	0.993	0.990	0.926	0.012	0.323	**0.712**	0.989	0.974	0.748	0.875	0.705	0.503	
LD-tSNE	0.886	0.952	0.841	0.227	0.341	0.636	0.949	0.939	0.619	0.570	0.404	0.761	

esc50

TF-PCA	0.650	0.627	0.450	0.699	0.434	0.542	0.954	0.927	0.502	0.508	0.354	0.995	
TF-tSNE	0.664	0.660	0.475	0.670	0.651	0.632	0.951	**0.948**	-0.07	-0.07	-0.05	2.153	
TF-UMAP	0.659	0.658	0.470	0.680	0.624	0.613	0.944	0.943	0.343	0.372	0.225	1.312	
G-AE	0.712	0.725	0.531	0.574	0.614	0.573	0.933	0.936	0.328	0.469	0.324	1.342	
G-VAE	0.638	0.636	0.452	0.723	0.589	0.588	0.938	0.939	0.357	0.391	0.266	1.284	
G-PCA	0.648	0.626	0.448	0.702	0.434	0.542	0.954	0.927	0.447	0.429	0.295	1.105	
G-tSNE	**0.740**	**0.750**	**0.554**	**0.518**	0.660	0.610	0.940	0.938	0.208	**0.886**	**0.697**	1.582	
G-UMAP	0.710	0.713	0.518	0.579	**0.669**	0.585	0.913	0.931	0.078	0.248	0.168	1.843	
C-tSNE	0.457	0.417	0.286	1.085	0.589	0.561	0.873	0.889	0.368	0.425	0.297	1.263	
C-UMAP	0.641	0.637	0.452	0.716	0.623	0.610	0.943	0.943	**0.528**	0.556	0.392	**0.943**	
D-tSNE	0.721	0.734	0.535	0.557	0.636	**0.642**	0.938	0.944	0.286	0.346	0.236	1.427	
PCD-tSNE	0.702	0.700	0.508	0.595	0.594	0.621	**0.961**	0.945	0.513	0.631	0.446	0.972	
LD-tSNE	0.722	0.726	0.530	0.555	0.638	0.630	0.951	0.948	0.397	0.661	0.477	1.204	

fashion

	$S_{Pearson}$	$S_{Spearman}$	$S_{Kendall}$	S_{Stress}	S_{NH}	S_{NP}	S_{Trust}	S_{Cont}	$T_{Pearson}$	$T_{Spearman}$	$T_{Kendall}$	T_{Stress}	
TF-PCA	0.431	**0.500**	**0.364**	1.136	0.837	0.536	0.880	0.852	0.808	0.915	0.739	0.383	
TF-tSNE	-0.06	0.172	0.132	2.126	0.948	0.611	0.880	0.881	-0.08	-0.04	-0.03	2.163	
TF-UMAP	-0.24	0.117	0.103	2.494	0.957	0.589	0.878	0.883	0.553	0.406	0.268	0.892	
G-AE	0.226	0.361	0.264	1.546	0.964	0.581	0.882	0.889	0.679	0.880	0.699	0.640	
G-VAE	0.264	0.384	0.282	1.470	0.978	0.585	**0.882**	0.887	0.668	0.917	0.741	0.662	
G-PCA	**0.443**	0.491	0.362	**1.112**	0.847	0.539	0.880	0.852	0.841	0.945	0.790	0.317	
G-tSNE	0.218	0.396	0.293	1.563	0.995	0.567	0.880	0.874	-0.07	-0.17	-0.15	2.153	
G-UMAP	0.220	0.368	0.268	1.559	0.997	0.593	0.877	0.885	0.063	-0.35	-0.24	1.873	
C-tSNE	-0.14	-0.09	-0.06	2.285	0.672	0.422	0.760	0.750	-0.04	0.191	0.170	2.080	
C-UMAP	-0.30	-0.06	-0.05	2.604	0.953	0.583	0.879	0.881	0.737	0.420	0.293	0.524	
D-tSNE	0.152	0.279	0.199	1.695	**0.998**	**0.625**	0.878	**0.892**	0.811	0.868	0.694	0.376	
PCD-tSNE	0.442	0.490	0.361	1.114	0.856	0.549	0.882	0.856	**0.844**	**0.948**	**0.797**	**0.311**	
LD-tSNE	-0.09	0.082	0.063	2.199	0.969	0.611	0.880	0.884	0.652	0.535	0.366	0.694	
TF-PCA	0.697	**0.842**	0.674	0.605	0.230	0.453	0.935	0.860	0.151	0.312	0.223	1.696	
TF-tSNE	0.468	0.582	0.413	1.063	0.214	0.449	0.923	0.861	0.231	0.255	0.175	1.537	
TF-UMAP	0.494	0.666	0.499	1.010	0.185	0.434	0.908	0.818	0.009	0.007	0.005	1.981	
G-AE	0.630	0.795	0.630	0.738	0.239	0.454	0.929	0.854	0.579	0.524	0.362	0.841	
G-VAE	0.558	0.745	0.564	0.882	**0.245**	0.317	0.799	0.792	0.619	**0.582**	**0.408**	0.760	
G-PCA	0.629	0.794	0.628	0.741	0.233	0.452	0.929	0.853	0.565	0.501	0.344	0.869	
G-tSNE	**0.719**	0.724	0.549	**0.561**	0.216	0.436	0.898	0.837	0.474	0.565	0.403	1.050	
G-UMAP	0.509	0.634	0.465	0.981	0.218	0.385	0.869	0.812	0.279	0.392	0.266	1.440	
C-tSNE	0.524	0.738	0.586	0.951	0.235	**0.489**	**0.941**	**0.880**	0.535	0.390	0.269	0.928	
C-UMAP	0.475	0.648	0.482	1.048	0.186	0.421	0.902	0.810	-0.07	-0.03	-0.02	2.153	
D-tSNE	0.448	0.619	0.448	1.103	0.099	0.101	0.509	0.531	-0.05	-0.03	-0.02	2.109	
PCD-tSNE	0.627	0.802	0.640	0.744	0.229	0.480	0.939	0.867	0.532	0.462	0.316	0.935	
LD-tSNE	0.688	0.840	**0.676**	0.622	0.235	0.462	0.936	0.877	**0.628**	0.543	0.382	**0.742**	
TF-PCA	**0.998**	**0.980**	**0.917**	**0.002**	**0.729**	0.568	0.952	0.957	0.370	0.526	0.369	1.259	
TF-tSNE	0.286	0.688	0.508	1.426	0.703	**0.658**	0.977	**0.971**	0.220	-0.04	-0.03	1.559	
TF-UMAP	0.668	0.764	0.560	0.663	0.685	0.563	0.949	0.941	0.174	0.058	0.037	1.650	
G-AE	0.987	0.977	0.893	0.024	0.709	0.567	0.955	0.954	0.832	0.971	0.870	0.335	
G-VAE	0.968	0.967	0.864	0.063	0.716	0.557	0.957	0.953	**0.861**	**0.975**	**0.878**	**0.277**	
G-PCA	0.998	0.977	0.909	0.002	0.720	0.563	0.951	0.954	0.855	0.965	0.864	0.288	
G-tSNE	0.556	0.594	0.431	0.887	0.496	0.406	0.782	0.822	0.547	0.528	0.435	0.905	
G-UMAP	0.472	0.484	0.349	1.054	0.436	0.344	0.648	0.779	0.262	0.645	0.467	1.475	
C-tSNE	0.723	0.796	0.606	0.553	0.713	0.651	**0.977**	0.970	0.276	0.289	0.198	1.447	
C-UMAP	0.740	0.802	0.604	0.519	0.686	0.555	0.951	0.947	0.223	0.227	0.155	1.552	
D-tSNE	0.588	0.684	0.494	0.822	0.187	0.180	0.670	0.657	-0.03	-0.05	-0.03	2.064	
PCD-tSNE	0.997	0.964	0.884	0.004	0.717	0.624	0.976	0.966	0.687	0.505	0.360	0.624	
LD-tSNE	0.825	0.842	0.655	0.349	0.706	0.607	0.969	0.962	0.242	0.699	0.492	1.515	
TF-PCA	0.608	0.685	0.500	0.783	0.479	0.250	0.774	0.704	0.376	0.739	0.589	1.247	
TF-tSNE	0.092	0.243	0.178	1.815	0.575	0.318	0.787	0.737	0.117	0.071	0.054	1.764	
TF-UMAP	0.116	0.344	0.246	1.767	0.539	0.295	0.743	0.708	0.334	0.331	0.253	1.330	
G-AE	0.560	**0.769**	**0.595**	0.879	**0.613**	0.259	0.754	0.715	0.277	0.938	0.827	1.445	
G-VAE	0.675	0.769	0.586	0.649	0.595	0.270	0.728	0.726	0.438	0.953	0.857	1.122	
G-PCA	0.655	0.738	0.544	0.689	0.482	0.213	0.671	0.672	**0.856**	**0.970**	**0.919**	**0.287**	
G-tSNE	0.602	0.622	0.443	0.795	0.276	0.128	0.533	0.550	0.304	0.906	0.829	1.390	
G-UMAP	**0.696**	0.759	0.566	**0.606**	0.203	0.119	0.539	0.538	0.277	0.647	0.511	1.444	
C-tSNE	0.592	0.713	0.521	0.815	0.537	0.287	0.753	0.705	0.434	0.547	0.422	1.131	
C-UMAP	0.089	0.341	0.245	1.821	0.541	0.291	0.744	0.705	0.404	0.459	0.349	1.190	
D-tSNE	0.363	0.499	0.375	1.272	0.575	**0.319**	**0.840**	**0.773**	0.436	0.625	0.490	1.127	
PCD-tSNE	0.642	0.740	0.541	0.714	0.507	0.242	0.742	0.688	0.763	0.762	0.625	0.472	
LD-tSNE	0.400	0.528	0.376	1.198	0.515	0.275	0.766	0.700	0.435	0.648	0.500	1.129	

gaussians

nnset

qtables

quickdraw

low quality — high quality

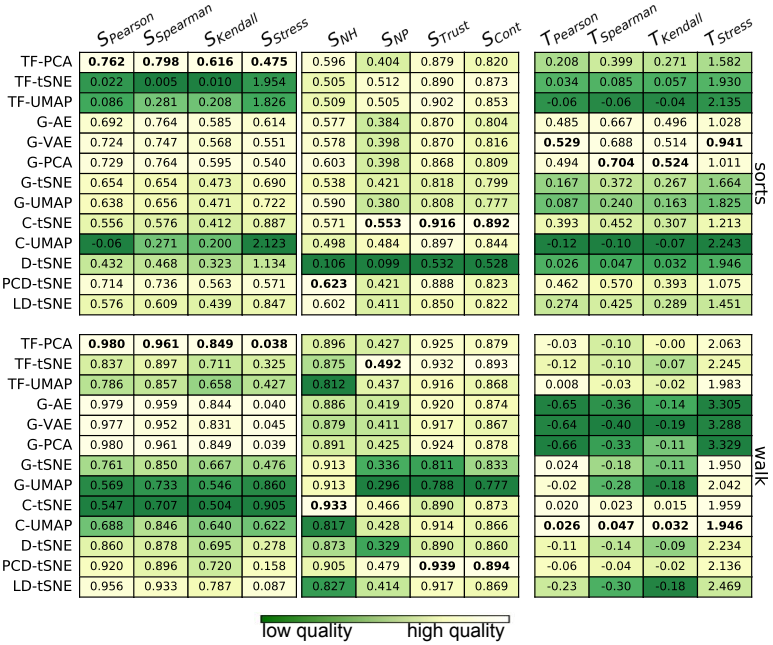


Table 9: Unaggregated metric results.

BIBLIOGRAPHY

W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visual methods for analyzing time-oriented data. *IEEE TVCG*, 14(1):47–60, 2008.

H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM*, 17(4):589–602, 1970. ISSN 0004-5411.

G. Albuquerque, M. Eisemann, and M. Magnor. Perception-based visual quality measures. In *Proc. IEEE VAST*, pages 11–18, 2011.

M. Ali, M. W. Jones, X. Xie, and M. Williams. TimeCluster: Dimension reduction applied to temporal data for visual analytics. *Visual Computer*, 35(6-8):1013–1026, 2019.

D. Archambault, H. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE TVCG*, 17(4):539–552, 2011.

M. Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 10(7–9):1304–1330, 2007.

M. Aupetit and M. Sedlmair. SepMe: 2002 New visual separation measures. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–8, 2016.

B. Bach, C. Shi, and N. Heulot. Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE TVCG*, 22(1):559–568, 2016.

P. Bain, L. Findley, P. Atchison, M. Behari, M. Vidailhet, M. Gresty, J. Rothwell, P. Thompson, and C. Marsden. Assessing tremor severity. *J Neurol Neurosurg Psychiatry*, 56(8):868–873, 1993.

D. H. Ballard. Modular learning in neural networks. *AAAI*, pages 279–284, 1987.

M. Balzer and O. Deussen. Voronoi treemaps. In *Proc. IEEE InfoVis*, pages 49–56, 2005.

M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proc. ACM SOFTVIS*, pages 165–172, 2005.

G. D. Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry*, 7(5):303–325, 1997.

E. Becht, L. McInnes, J. Healy, C. A. Dutertre, I. W. H. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell. Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology*, 37(1):38–44, 2019.

R. A. Becker, W. S. Cleveland, and M. J. Shyu. The visual design and control of trellis display. *JCGS*, 5(2):123–155, 1996.

B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM TOG*, 21(4):833–854, 2002.

E. Beghi, V. Regio, A. Papantonio, A. Bentivoglio, A. Fasano, D. Fogli, L. Giordano, R. Piolti, G. Rinaldi, P. Simone, L. Specchio, P. Tonali, P. Torelli, M. Zarrelli, and P. Messina. Reliability of clinical diagnosis of dystonia. *Neuroepidemiology*, 43:213–219, 11 2014.

Y. Bengio, J. F. Paiement, P. Vincent, O. Delalleau, N. L. Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and spectral clustering. In *Proc. NIPS*, pages 177–184, 2003.

M. D. Berg, B. Speckmann, and V. van der Weele. Treemaps with bounded aspect ratio. *Computational Geometry*, 47(6):683–693, 2014.

J. Bernard, N. Wilhelm, and M. Scherer. TimeSeriesPaths: Projection-based explorative analysis of multivariate time series data. *Journal of WSCG*, pages 97–106, 2012.

I. Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115–147, 1987.

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

S. A. Boorman and D. C. Oliviera. Metrics on spaces of finite trees. In *Journal of Mathematical Psychology*, volume 10, pages 26–59, 1973.

A. Boytsov, F. Fouquet, T. Hartmann, and Y. L. Traon. Visualizing and exploring dynamic high-dimensional datasets with LION-tSNE. *ArXiv*, abs/1708.04983, 2017.

M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe. A comparative evaluation of animation and small multiples for trend visualization on mobile phones. *IEEE TVCG*, PP, 2019.

N. Brich, C. Schulz, J. Peter, W. Klingert, M. Schenk, D. Weiskopf, and M. Krone. Visual analysis of multivariate intensive care surveillance data. In *Proc. Eurographics Workshop on Visual Computing for Biology and Medicine*, 2020.

M. Bruls, K. Huizing, , and J. J. V. Wijk. Squarified treemaps. In *Proc. VisSym*, pages 33–42. Springer, 2000.

K. Buchin, D. Eppstein, M. Löffler, M. Nöllenburg, and R. I. Silveira. Adjacency-preserving spatial treemaps. In *Algorithms and Data Structures*, pages 159–170, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

A. Buja, D. Cook, and D. F. Swayne. Interactive high-dimensional data visualization. *JCGS*, 5(1):78–99, 1996.

A. Buja, D. F. Swayne, M. L. Littman, N. Dean, H. Hofmann, and L. Chen. Data visualization with multidimensional scaling. *JCGS*, 17(2):444–472, 2008.

K. Bunte, M. Biehl, and B. Hammer. A general framework for dimensionality reducing data visualization mapping. *Neural Computation*, 24(3):771–804, 2012.

S. Card, B. Suh, B. A. Pendleton, B. Heer, and J. W. Bodnar. Time tree: Exploring time changing hierarchies. In *Proc. Symposium on Visual Analytics Science and Technology*, pages 3–10, 2006.

Y. Chen, X. Du, and X. Yuan. Ordered small multiple treemaps for visualizing time-varying hierarchical pesticide residue data. *Visual Computer*, 33(6):1073–1084, 2017.

F. Chollet et al. Keras. https://keras.io, 2015.

J. Clark. Multi-level pie charts, 2006. https://neoformix.com/2006/MultiLevelPieChart.html.

A. K. Cline and I. S. Dhillon. *Computation of the Singular Value Decomposition.* CRC Press, 2006.

B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. van Wijk, and A. van Deursen. Understanding execution traces using massive sequence and circular bundle views. In *Proc. IEEE ICPC*, pages 271–280, 2007.

CSV. CVS - Concurrent Versions System, 2018. https://www.nongnu.org/cvs/.

J. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *JMLR*, 16:2859–2900, 2015.

I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, 1990.

V. De Silva and J. B. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical Report 6, Stanford University, 2004.

V. De Silva and J. B. Tenenbaum. Selecting landmark points for sparse manifold learning. *Advances in Neural Information Processing Systems*, pages 1241–1248, 2005.

G. Defazio, G. Abbruzzese, P. Livrea, and A. Berardelli. Epidemiology of primary dystonia. *Lancet Neurol*, 11(3):673–678, 2004.

S. Diehl. *Software Visualization – Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007.

S. Duarte, F. Sikanski, F. Fatore, S. Fadel, and F. Paulovich. Nmap: A novel neighborhood preservation space-filling algorithm. *IEEE TVCG*, 20(12):2063–2071, 2014.

H. Eggink, D. Kremer, O. Brouwer, M. Contarino, M. van Egmond, A. Elema, K. Folmer, J. van Hoorn, L. van de Pol, V. Roelfsema, and M. Tijssen. Spasticity, dyskinesia and ataxia in cerebral palsy: Are we sure we can differentiate them? *European Journal of Paediatric Neurology*, 21(5):703–706, September 2017. ISSN 1090-3798.

A. Eklund. Beeswarm: The bee swarm plot, an alternative to stripchart, 2012. R package version 0.1 5.

B. Engdahl. Ordered and unordered treemap algorithms and their applications on handheld devices, 2005. MSc thesis, Dept. of CS, Stockholm Royal Institute of Technology.

D. Engel, L. Hüttenberger, and B. Hamann. A survey of dimension reduction methods for high-dimensional data analysis and visualization. In *Proc. IRTG Workshop*, volume 27, pages 135–149, 2012.

D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek. Area-universal and constrained rectangular layouts. *SIAM Journal on Computing*, 41 (3):537–564, 2012.

M. Espadoto, E. Vernier, and A. Telea. Selecting and Sharing Multidimensional Projection Algorithms: A Practical View. In *VisGap - The Gap between Visualization Research and Visualization Software*. The Eurographics Association, 2020a. ISBN 978-3-03868-125-0.

M. Espadoto, R. Martins, A. Kerren, N. Hirata, and A. Telea. Towards a quantitative survey of dimension reduction techniques. *IEEE TVCG*, pages 1–1, 2019.

M. Espadoto, N. S. T. Hirata, and A. C. Telea. Deep learning multidimensional projections. *Information Visualization*, 19(3):247–269, 2020b.

D. Fisher and A. Sud. Animated, dynamic Voronoi treemaps. In *Proc. EuroVis – posters*. Eurographics, 2010.

I. K. Fodor. A survey of dimension reduction techniques. *US Dept. of Energy, Lawrence Livermore National Labs*, 2002. Tech. report UCRL-ID-148494.

A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of the DIMACS International Workshop on Graph Drawing*, GD '94, pages 388–403. Springer-Verlag, 1995.

T. Fujiwara, J. Chou, S. Shilpika, P. Xu, L. Ren, and K. Ma. An incremental dimensionality reduction method for visualizing streaming multidimensional data. *IEEE TVCG*, pages 1–1, 2019.

T. Fujiwara, Shilpika, N. Sakamoto, J. Nonaka, K. Yamamoto, and K. L. Ma. A visual analytics framework for reviewing multivariate time-series data with dimensionality reduction. *IEEE TVCG*, 2020.

T. Fujiwara, J. K. Li, M. Mubarak, C. Ross, C. D. Carothers, R. B. Ross, and K. L. Ma. A visual analytics system for optimizing the performance of large-scale networks in supercomputing systems. *Visual Informatics*, 2(1):98–110, 2018.

F. J. García-fernández, M. Verleysen, J. a. Lee, and I. Díaz. Stability comparison of dimensionality reduction techniques attending to data and parameter variations. *EuroVis Workshop on Visual Analytics using Multidimensional Projections*, pages 2–6, 2013.

M. Ghoniem, M. Cornil, B. Broeksema, M. Stefas, and B. Otjacques. Weighted maps: Treemap visualization of geolocated quantitative data. In *Proc. Int'l Soc. for Optics and Photonics*, volume 9397, pages 1–15, 2015.

A. Gisbrecht and B. Hammer. Data visualization by nonlinear dimensionality reduction. *WIREs Data Mining Knowledge Discovery*, 5:51–73, 2015.

GIT. GIT source code management, 2018. https://git-scm.com/.

H. Gomide and A. Gualberto. caRtola, 2019. https://github.com/henriquepgomide/caRtola.

J. Görtler, C. Schulz, D. Weiskopf, and O. Deussen. Bubble treemaps for uncertainty visualization. *IEEE TVCG*, 24:719–728, 2018.

D. Gotz. Dynamic Voronoi treemaps: A visualization technique for time-varying hierarchical data. *Computer Science – Research and Development*, 18:132–141, 2011. IBM Research Report RC25132 (W1103-173).

M. Graham and J. Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

J. M. Graving and I. D. Couzin. Vae-sne: a deep generative model for simultaneous dimensionality reduction and clustering. *bioRxiv*, 2020.

C. Grillenzoni and M. Fornaciari. On-line peak detection in medical time series with adaptive regression methods. *Econometrics and Statistics*, 10:134 – 150, 2019.

P. Grosse, R. Guerrini, L. Parmeggiani, P. Bonanni, A. Pogosyan, and P. Brown. Abnormal corticomuscular and intermuscular coupling in high-frequency rhythmic myoclonus. *Brain*, 126(2):326–342, 2003.

P. Grosse, M. Edwards, M. Tijssen, A. Schrag, A. J. Lees, K. Bhatia, and P. Brown. Patterns of emg–emg coherence in limb dystonia. *Movement disorders*, 19(7):758–769, 2004.

J. Guerra-Gómez, M. Pack, C. Plaisant, and B. Shneiderman. Visualizing change over time using dynamic hierarchies: TreeVersity2 and the StemView. *IEEE TVCG*, 19(12):2566–2575, 2013.

L. Gupta, D. Molfese, R. Tammana, and P. Simos. Nonlinear alignment and averaging for estimating the evoked potential. *IEEE Transactions on Biomedical Engineering*, 43(4):348–356, 1996.

S. Hachul and M. Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In *Proceedings of the 13th International Conference on Graph Drawing*, GD'05, pages 235–250. Springer-Verlag, 2006.

S. Hahn and J. Döllner. Hybrid-treemap layouting. In *Proc. EuroVis (short papers)*, 2017.

S. Hahn, J. Trümper, D. Moritz, and J. Döllner. Visualization of varying hierarchies by stable layout of Voronoi treemaps. In *Proc. IEEE IVAPP*, pages 50–58, 2014.

S. Hahn. Comparing the layout stability of treemap algorithms. *Proc. HPI research school on service-oriented systems engineering*, 95:71–79, 2015.

S. Hahn, J. Bethge, and J. Döllner. Relative direction change: A topology-based metric for layout stability in treemaps. In *International Conference on Information Visualization Theory and Applications*, pages 88–95, 2017.

D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *10th International Symposium on Graph Drawing*, GD '02, pages 207–219. Springer-Verlag, 2002.

S. Haroz, R. Kosara, and S. Franconeri. The connected scatterplot for presenting paired time series. *IEEE TVCG*, 22:1–1, 2015.

F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19, 2016.

R. van Hees and J. Hage. Stable and predictable Voronoi treemaps for software quality monitoring. *Inf Soft Technol*, 87(C):242–258, 2017.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

P. Hoffman and G. Grinstein. A survey of visualizations for high-dimensional data mining. *Information Visualization in Data Mining and Knowledge Discovery*, 104:47–82, 2002.

D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG*, 12(5):741–748, 2006.

Y. Hu, S. Wu, S. Xia, J. Fu, and W. Chen. Motion track: Visualizing variations of human motion data. *Proc. IEEE PacificVis*, pages 153–160, 2010.

C. Hurter, O. Ersoy, and A. Telea. Smooth bundling of large streaming and sequence graphs. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pages 41–48, 2013.

A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proc. IEEE VIS*, pages 361–378, 1990.

D. Jäckle, F. Fischer, T. Schreck, and D. A. Keim. Temporal MDS plots for analysis of multivariate data. *IEEE TVCG*, 22(1):141–150, 2016.

P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Local affine multidimensional projection. *IEEE TVCG*, 17(12):2563–2571, 2011.

I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The Quick, Draw! - A.I. Experiment. https://quickdraw.withgoogle.com/, 2016.

H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Software: Evolution and Process*, 19(2):77–131, 2003.

J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE TVCG*, 19(3):495–513, 2013.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

T. Kohonen, M. R. Schroeder, and T. S. Huang. *Self-Organizing Maps*. Springer-Verlag, 3rd edition, 2001.

N. Kong, J. Heer, and M. Agrawala. Perceptual guidelines for creating rectangular treemaps. *IEEE TVCG*, 16(6):990–998, 2010.

W. Köpp and T. Weinkauf. Temporal treemaps: Static visualization of evolving trees. *IEEE TVCG*, 25(1):534–543, 2019.

G. Kramer, A. Van der Stouwe, N. Maurits, M. Tijssen, and J. Elting. Wavelet coherence analysis: a new approach to distinguish organic and functional tremor types. *Clinical Neurophysiology*, 129(1):13–20, 2018.

A. A. Krapl. The time-varying diversifiability of corporate foreign exchange exposure. *Journal of Corporate Finance*, page 101506, 2019.

D. Kressner. Numerical methods for general and structured eigenvalue problems. *Lecture Notes in Computational Science and Engineering*, 46, 2005.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009.

J. F. Kruiger, A. C. Telea, and C. Hurter. Projection navigation in extremely large datasets (PNIELD). In *Proc. EuroVis: Posters*, pages 109–111. Eurographics Association, 2017a.

J. Kruiger, A. Hassoumi, H. J. Schulz, A. Telea, and C. Hurter. Multidimensional data exploration by explicitly controlled animation. *Informatics*, 4(26), 2017b.

J. B. Kruskal and J. M. Landwehr. Icicle Plots: Better Displays for Hierarchical Clustering. *The American Statistician*, 37(2):162–168, 1983.

M. K. Kuhner and J. Yamato. Practical performance of tree comparison metrics. *Systematic Biology*, 64(2):205–214, 2015.

T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J. D. Fekete, and D. W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *CGF*, 30(6):1719–1749, 2011.

M. Lanza and S. Ducasse. Polymetric views: A lightweight visual approach to reverse engineering. *IEEE Trans Soft Eng*, 29(9):782–795, 2003.

M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer, 2006.

Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

J. A. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7-9):1431–1443, 2009.

S. Lespinats and M. Aupetit. CheckViz: Sanity check and topological clues for linear and nonlinear mappings. *CGF*, 30(1):113–125, 2011.

G. Li, Y. Zhang, Y. Dong, J. Liang, J. Zhang, J. Wang, M. J. McGuffin, and X. Yuan. BarcodeTree: Scalable comparison of multiple hierarchies. *IEEE TVCG*, 26(1):1022–1032, 2019.

G. C. Linderman and S. Steinerberger. Clustering with T-SNE, provably. *arXiv*, 1:1–15, 2017.

G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger. Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data. *Nature Methods*, 16(3):243–245, 2019.

S. Liu, D. Maljovec, B. Wang, P. T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE TVCG*, 23 (3):1249–1268, 2017.

L. Lu, S. Fan, M. Huang, W. Huang, and R. Yang. Golden rectangle treemap. *Journal of Physics: Conference Series*, 787(1), 2017.

W. Lueks, A. Gisbrecht, and B. Hammer. Visualizing the quality of dimensionality reduction. *Neurocomputing*, 112:109–123, 2013.

J. Lukasczyk, G. Weber, R. Maciejewski, C. Garth, and H. Leitte. Nested tracking graphs. *Computer Graphics Forum*, 36(3):12–22, 2017.

L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 9:2579–2605, 2008.

Y. Mao, J. V. Dillon, and G. Lebanon. Sequential document visualization. *IEEE TVGC*, 13(6):1208–1215, 2007.

R. M. Martins, D. B. Coimbra, R. Minghim, and A. C. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *CG*, 41(1):26–42, 2014.

R. M. Martins, R. Minghim, and A. C. Telea. Explaining neighborhood preservation for multidimensional projections. *CGVC*, 2015.

J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *ICANN*, pages 52–59, 2011.

L. McInnes, J. Healy, N. Saul, and L. Großberger. UMAP: Uniform manifold approximation and projection. *JOSS*, 3(29):861, 2018.

L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz. A systematic literature review of software visualization evaluation. *J Syst Softw*, 144: 165–180, 2018.

M. Mihalec, J. Slavič, and M. Boltežar. Synchrosqueezed wavelet transform for damping identification. *Mechanical Systems and Signal Processing*, 80:324–334, 2016. ISSN 0888-3270.

H. A. Müller and K. Klashinsky. Rigi: A system for programming-in-the-large. In *Proc. IEEE ICSE*, pages 80–86, 1988.

H. Nagamochi and Y. Abe. An approximation algorithm for dissecting a rectangle into rectangles with specified areas. *Discrete Applied Mathematics*, 155(4):523–537, 2007.

T. T. Neves, S. G. Fadel, G. M. Hilasaca, F. M. Fatore, and F. V. Paulovich. UPDis: A user-assisted projection technique for distance information. *Information Visualization*, 17(4):269–281, 2018.

T. T. Neves, R. M. Martins, D. B. Coimbra, K. Kucher, A. Kerren, and F. Paulovich. Xtreaming: An incremental multidimensional projection technique and its application to streaming data. *CoRR*, 2020.

M. Nguyen, S. Purushotham, H. To, and C. Shahabi. m-TSNE: A framework for visualizing high-dimensional multivariate time series. *ArXiv*, abs/1708.07942, 2017.

S. Nijmeijer, E. De Bruijn, P. Forbes, D. Kamphuis, R. Happee, J. Koelman, and M. Tijssen. Emg coherence and spectral analysis in cervical dystonia: Discriminative tools to identify dystonic muscles? *Journal of the neurological sciences*, 347(1-2):167–173, 2014.

L. G. Nonato and M. Aupetit. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG*, 25(8):2650–2673, 2019.

E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1087, 1962.

F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14 (3):564–575, 2008.

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

E. Pekalska, D. de Ridder, R. P. Duin, and M. A. Kraaijveld. A new method of generalizing Sammon mapping with application to algorithm speed-up. In *Proc. ASCI*, pages 221–228, 1999.

N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. *Computer Graphics Forum*, 35(3):21–30, 2016.

K. J. Piczak. ESC: Dataset for Environmental Sound Classification. In *Proc. ACM MM*, pages 1015–1018, 2015.

P. G. Poličar, M. Stražar, and B. Zupan. Embedding to reference t-sne space addresses batch effects in single-cell classification. In *Discovery Science*, pages 246–260. Springer, 2019.

G. Pölzlbauer. Survey and comparison of quality measures for self-organizing maps. In *Proc. Workshop on Data Analysis (WDA)*, pages 67–82, 2004.

R. Rao and S. K. Card. The table lens. *Proc. CHI*, page 222, 1994.

P. Rauber, A. Falcão, and A. Telea. Projections as visual aids for classification system design. *Information Visualization*, 17:282–305, 2017a.

P. E. Rauber, A. X. Falcão, and A. C. Telea. Visualizing time-dependent data using dynamic t-SNE. *EuroVis*, 2016.

P. E. Rauber, S. G. Fadel, A. X. Falcão, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE TVCG*, 23(1):101–110, 2017b.

D. A. Ross, J. Lim, R. S. Lin, , and M. H. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77 (1-3):125–141, 2008.

G. van Rossum. The Python programming language, 2017. https://github.com/python/cpython.

W. Scheibel, M. Trapp, D. Limberger, and J. Döllner. A taxonomy of treemap visualization techniques. In *Proc. International Conference on Information Visualization Theory and Applications*, 2020.

W. Scheibel, C. Weyand, and J. Döllner. EvoCells: A treemap layout algorithm for evolving tree data. In *Proc. International Conference on Information Visualization Theory and Applications*, pages 273–280, 2018.

T. Schreck, T. von Landesberger, and S. Bremm. Techniques for precision-based visual analysis of projected data. *Information Visualization*, 9(3):181–193, 2010.

H. J. Schulz. Treevis.net: A tree visualization reference. *IEEE CG&A*, 31 (6):11–15, 2011.

H. J. Schulz, S. Hadlak, and H. Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE TVCG*, 17(4):393–411, 2011.

SciTools. Understand static code analysis tool, 2017. https://scitools.com.

M. Sedlmair and M. Aupetit. Data-driven evaluation of visual quality measures. *CGF*, 34(3):545–559, 2015.

M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE TVCG*, pages 2634–2643, 2013.

M. Sensalire, P. Ogao, and A. Telea. Evaluation of software visualization tools: Lessons learned. In *Proc. IEEE VISSOFT*, 2009.

A. Seriai, O. Benomar, B. Cerat, and H. Sahraoui. Validation of software visualization tools: A systematic mapping study. In *Proc. IEEE VISSOFT*, 2014.

J. Shi and C. Tomasi. Good features to track. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

B. Shneiderman and C. Plaisant. Treemaps for space-constrained visualization of hierarchies, 2017.

B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *Proc. IEEE InfoVis*, pages 73–80, 2001.

B. Shneiderman. Tree visualization with tree-maps: 2-D space-filling approach. *ACM TOG*, 11(92), 1992.

R. da Silva, E. Vernier, P. Rauber, J. Comba, R. Minghim, and A. Telea. Metric evolution maps: Multidimensional attribute-driven exploration of software repositories. In *Proc. Vision, Modeling, and Visualization (VMV)*, pages 54–62. Eurographics, 2016.

V. Silva and J. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. *Adv Neural Inf Process Syst*, 15, 2003.

M. Sips, B. Neubert, J. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. *CGF*, 28(3):831–838, 2009.

K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

M. Sondag, B. Speckmann, and K. Verbeek. Stable treemaps via local moves. *IEEE TVCG*, 24(1):729–738, 2017.

C. Sorzano, J. Vargas, and A. Pascual-Montano. A survey of dimensionality reduction techniques. *ArXiv*, abs/1403.2877, 2014.

P. Steinhardt. *libgit2* API for Git repository management, 2018. https://libgit2.github.com.

A. van der Stouwe, B. A. Conway, J. Elting, M. Tijssen, and N. M. Maurits. Usefulness of intermuscular coherence and cumulant analysis in the diagnosis of postural tremor. *Clinical Neurophysiology*, 126(8):1564–1569, 2015.

Subversion. Apache Subversion: Enterprise-class centralized version control for the masses, 2018. https://subversion.apache.org/.

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

S. Tak and A. Cockburn. Enhanced spatial stability with Hilbert and Moore treemaps. *IEEE TVCG*, 19(1):141–148, 2013.

A. Tatu, P. Bak, E. Bertini, D. Keim, and J. Schneidewind. Visual quality metrics and human perception: An initial study on 2D projections of large multidimensional data. In *Proc. AVI*, pages 49–56, 2010.

A. Telea. Combining extended table lens and treemap techniques for visualizing tabular data. In *Proc. VGTC Conference on Visualization*, pages 120–127, 2006.

A. Telea, A. Maccari, and C. Riva. An open toolkit for prototyping reverse engineering visualizations. In *Proc. Data Visualisation (VisSym)*, pages 241–249, 2002.

A. Telea, H. Hoogendorp, O. Ersoy, and D. Reniers. Extraction and visualization of call dependencies for large C/C++ code bases: A comparative study. In *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 81–88, 2009.

G. Teo, Y. Bin Zhang, C. Vogel, and H. Choi. PECAplus: statistical analysis of time-dependent regulatory changes in dynamic single-omics and dual-omics experiments. *npj Systems Biology and Applications*, 4 (1):3, 2017.

The Authors. Dynamic treemap for software evolution visualization benchmark, 2017. https://github.com/vissoft18/treemaps.

The Authors. GIT - Dynamic Treemap Benchmark, 2018. https://github.com/sibgrapi18/treemaps.

The Authors. Additional resources repository, 2019. https://eduardovernier.github.io/dynamic-projections/.

The Authors. Treemap resources. https://eduardovernier.github.io/dynamic-treemap-resources-eurovis, 2020a.

The Authors. Additional resources repository, 2020b. https://eduardovernier.github.io/guided-dynamic-projections-resources/.

M. Tijssen, J. Marsden, and P. Brown. Frequency analysis of emg activity in patients with idiopathic torticollis. *Brain*, 123(4):677–686, 2000.

Y. Tu and H. W. Shen. Visualizing changes of hierarchical data using treemaps. *IEEE TVCG*, 13(6):1286–1293, 2007.

T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision*, 3 (3):177–280, 2007.

URL. ATP Tennis Rankings. https://github.com/JeffSackmann/tennis_atp, accessed 03-07-2018a.

URL. USGS Earthquakes. https://earthquake.usgs.gov/earthquakes/browse/stats.php, accessed 03-07-2018b.

URL. Worldbank indicators. https://data.worldbank.org/indicator/, accessed 04-07-2018.

URL. The Movie Database. www.themoviedb.org, accessed 10-02-2018.

URL. UN Comtrade Database. https://comtrade.un.org, accessed 15-02-2017a.

URL. Scitools. https://scitools.com, accessed 15-02-2017b.

URL. Github. https://github.com, accessed 16-07-2018.

URL. Meertens Instituut, KNAW nederlandse voornamenbank. https://www.meertens.knaw.nl/nvb, accessed 30-05-2016.

L. van der Maaten. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2015.

L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

L. Van Der Maaten, E. Postma, and J. Van den Herik. Dimensionality reduction: A comparative review. *JMLR*, 10:66–71, 2009.

S. van der Salm, R. de Haan, D. Cath, A. F. van Rootselaar, and M. Tijssen. The eye of the beholder: Inter-rater agreement among experts on psychogenic jerky movement disorders. *Journal of Neurology, Neurosurgery and Psychiatry*, 84(7):742–747, 2013.

A. van der Stouwe, I. Tuitert, I. Giotis, J. Calon, R. Gannamani, J. Dalenberg, S. van der Veen, M. Klamer, A. Telea, and M. Tijssen. The next move in movement disorders (nemo): developing a computer aided classification tool for hyperkinetic movement disorders. *BMJ Open [Accepted]*, 2021.

S. van der Veen, M. Klamer, J. Elting, J. Koelman, A. van der Stouwe, and M. Tijssen. The diagnostic value of clinical neurophysiology in hyperkinetic movement disorders: A systematic review. *Parkinsonism and Related Disorders*, 89:176–185, 2021. ISSN 1353-8020.

J. Venna and S. Kaski. Visualizing gene interaction graphs with local multidimensional scaling. In *Proc. ESANN*, pages 557–562, 2006.

E. Vernier, J. Comba, and A. Telea. Quantitative comparison of treemap techniques for time-dependent hierarchies. In *Proc. EuroVis (posters)*, 2017.

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*, pages 158–165, 2018a.

E. Vernier, J. Comba, and A. Telea. Quantitative comparison of dynamic treemaps for software evolution visualization. In *IEEE Conference on Software Visualization*, pages 96–106, 2018b.

E. Vernier, R. Garcia, I. da Silva, J. Comba, and A. Telea. Quantitative evaluation of time-dependent multidimensional projection techniques. *Computer Graphics Forum*, 39(3):241–252, 2020a.

E. Vernier, M. Sondag, J. Comba, B. Speckmann, A. Telea, and K. Verbeek. Quantitative comparison of time-dependent treemaps. *Computer Graphics Forum*, 39(3):393–404, 2020b.

E. Vernier, J. Comba, and A. Telea. Guided stable dynamic projections. *Computer Graphics Forum*, 2021. ISSN 1467-8659.

M. Vladymyrov and M. Á. Carreira-Perpiñán. Locally linear landmarks for large-scale manifold learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 256–271, 2013.

R. Vliegen, J. J. van Wijk, and E. J. van der Linden. Visualizing business data with generalized treemaps. *IEEE TVCG*, 12(5):789–796, 2006.

Y. Wang, K. Feng, X. Chu, J. Zhang, C. W. Fu, M. Sedlmair, X. Yu, and B. Chen. A Perception-Driven Approach to Supervised Dimensionality Reduction for Visualization. *IEEE TVCG*, 24(5):1828–1840, 2018.

M. O. Ward and Z. Guo. Visual exploration of time-series data with shape space projections. *CGF*, 30(3):701–710, 2011.

C. Watkins. Learning from delayed rewards, 1989. Ph.D. thesis, Cambridge University, UK.

M. Wattenberg. A note on space-filling visualizations and space-filling curves. In *Proc. IEEE InfoVis*, pages 181–186, 2005.

J. Wood and J. Dykes. Spatially ordered treemaps. *IEEE TVCG*, 14(6): 1348–1355, 2008.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

H. Yin. Nonlinear dimensionality reduction and data visualization: A review. *IJAC*, 4(3):294–303, 2007.

M. Zhou, Y. Cheng, N. Ye, and J. Tian. Effectiveness and efficiency of using different types of rectangular treemap as diagrams in cartography. In *International Cartographic Conference*, pages 187–206, 2017.

ACKNOWLEDGMENTS

I would like to thank my advisors Prof. Alexandru Telea and Prof. João Comba for all the support and guidance they so skillfully and kindly provided me during my PhD. I couldn't possibly imagine better companions for this journey.

I would like to thank my family for their love, their patience, their inspiration, and for always providing me all I've ever needed to succeed.

I would also like to thank all my friends both in Brazil and in the Netherlands. Krislen, Lianne, Alister, Samuel, Stefan, and Agathe, thank you for lifting me up whenever I was down.

This thesis was financed in part by CAPES (Finance Code 001, Scholarship Code 88882.345509/2019-01), CNPq (Process 304336/2019-0), and by the University of Groningen. My sincere thanks to these organizations for their support.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Final Version as of September 20, 2021 (`classicthesis`).