Visualizing Multidimensional Data Similarities
Improvements and Applications

Renato Rodrigues Oliveira da Silva

Cover: Hipothetical similarity tree of several concepts. Icons developed by Gregor Cresnar, Freepik and Zlatko Najdenovski from Flaticon (www.flaticon.com).

Visualizing Multidimensional Data Similarities
Improvements and Applications

Renato Rodrigues Oliveira da Silva
Thesis University of Groningen

university of
groningen

Visualizing multidimensional data similarities

Improvements and applications

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. E. Sterken
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Tuesday 18 October 2016 at 11.00 hours

by

Renato Rodrigues Oliveira da Silva

born on 5 November 1981
in Sao Jose dos Campos, Brazil

Supervisors
Prof. A. C. Telea
Prof. R. Minghim

Assessment Committee
Prof. M. Aiello
Prof. M. Biehl
Prof. L. Linsen
Prof. M. C. F. de Oliveira

Abstract

Multidimensional datasets are increasingly more prominent and important in data science and many application domains. Such datasets typically consist of a large set of observations, or data points, each which is described by several measurements, or dimensions. During the design of techniques and tools to process such datasets, a key component is to gather insights into their structure and patterns, a goal which is targeted by multidimensional visualization methods. Structures and patterns of high-dimensional data can be described, at a core level, by the notion of similarity of observations. Hence, to visualize such patterns, we need effective and efficient ways to depict similarity relations between a large number of observations, each having a potentially large number of dimensions. Within the realm of multidimensional visualization methods, two classes of techniques exist – projections and similarity trees – which effectively capture similarity patterns and also scale well to the number of observations and dimensions of the data. However, while such techniques show similarity patterns, understanding and interpreting these patterns in terms of the original data dimensions is still hard.

This thesis addresses the development of visual explanatory techniques for the easy interpretation of similarity patterns present in multidimensional projections and similarity trees, by several contributions. First, we propose methods that make the computation of similarity trees efficient for large datasets, and also allow their visual explanation on a multiscale, or several levels of detail. We also propose ways to construct simplified representations of similarity trees, thereby extending their visual scalability even further. Secondly, we propose methods for the visual explanation of multidimensional projections in terms of automatically detected groups of related observations which are also automatically annotated in terms of their similarity in the high-dimensional data space. We show next how these explanatory mechanisms can be adapted to handle both static and time-dependent multidimensional datasets. Our proposed techniques are designed to be easy to use, work nearly automatically, handle any types of quantitative multidimensional datasets and multidimensional projection techniques, and are demonstrated on a variety of real-world large datasets obtained from image collections, text archives, scientific measurements, and software engineeering.

Keywords:
Visualization, Visual analytics, Multidimensional data, Computer graphics

Samenvatting

Multidimensionele datasets zijn steeds prominenter and belangrijker in *data science* and een groot aantal toepassingsgebieden. Deze datasets bestaan in het algemeen uit een grote verzameling observaties, of datapunten, waarbij elk punt beschreven wordt door een aantal metingen of dimensies. Tijdens het ontwerpen van technieken en *tools* voor het verwerken van dergelijke datasets, een cruciale taak is het verzamelen van inzichten in de structuur en patronen van deze datasets. Deze taak is het onderwerp van multidimensionele visualisatiemethodes. Op een basisniveau kunnen structuren en patronen in hoogdimensionale data beschreven worden door het concept van similariteit van observaties. Voor het visualiseren van dergelijke patronen heeft men dus technieken nodig die similariteitsrelaties tussen veel observaties, elke met veel dimensies, afbeelden. In het context van multidimensionele visualisatiemethodes kent men twee klassen van technieken die similariteitspatronen effectief vangen en ook schaalbaar zijn in het aantal observaties en dimensies van de data: projecties en similariteitsbomen. Alhoewel deze technieken similariteitsrelaties kunnen afbeelden, het begrijpen en interpreteren van dergelijke patronen in termen van de oorspronkelijke datadimensies is uitdagend.

Dit proefschrift beschrijft de ontwikkeling van visueel uitlegtechnieken die het makkelijk interpreteren van similariteitspatronen in multidimensionele projecties en similariteitsbomen, mogelijk maken. Als eerste bijdrage presenteren wij methodes die het berekenen van similariteitsbomen efficiënt maakt voor grote datasets, en deze bomen ook visueel uitlegt op een multischaal manier, via verschillende niveaus van detail. We laten ook zien hoe versimpelde representaties van similariteitsbomen geproduceerd kunnen worden, waarbij hun visuele schaalbaarheid verder toeneemt. Als tweede bijdrage presenteren wij methodes voor het visueel uitleg van multidimensionale projecties door middel van automatisch gedetecteerde groepen van gerelateerde observaties die ook automatisch afgebeeld worden op basis van hun hoogdimensionale similariteit. We laten vervolgens zien hoe deze uitlegmechanismen aangepast kunnen worden voor het afhandelen van statische en ook tijdsafhankelijke datasets. Onze technieken zijn ontworpen voor gebruikersgemak, werken bijna volautomatisch, zijn te gebruiken met alle kwantitatieve multidimensionale datasets en projectietechnieken, en worden gedemonstreerd op een varieteit van realistische datasets uit beeldverzamelingen, tekstarchieven, wetenschapelijke metingen, en *software engineering*.

Trefwoorden:

Visualisatie, Visuele analyse, Multidimensionale gegevens, Computer graphiek

Resumo da tese

Conjuntos de dados multidimensionais são cada vez mais proeminentes e importantes em *data science* e muitos domínios de aplicação. Esses conjuntos de dados são tipicamente constituídos de um grande número de observações, ou objetos, cada qual descrito por várias medidas, ou dimensões. Durante o projeto de técnicas e ferramentas para processar tais dados, um dos focos principais é prover meios para análise e levantamento de hipóteses a partir das principais estruturas e padrões. Esse objetivo é peseguido por métodos de visualização multidimensional. Estruturas e padrões em dados multidimensionais podem ser descritos, em linhas gerais, pela noção de similaridade das observações. Portanto, para visualizar esses padrões, precisamos de meios efetivos e eficientes para retratar relações de similaridade dentre um grande número de observações, que potencialmente possuem um grande número de dimensões cada. No contexto dos métodos de visualização multidimensional, existem duas categorias de técnicas – projeções e árvores de similaridade – que efetivamente capturam padrões de similaridade e oferecem boa escalabilidade, tanto para o número de observações e quanto de dimensões. No entanto, embora essas técnicas exibam padrões de similaridade, o entendimento e interpretação desses padrões, em termos das dimensões originais dos dados, ainda é difícil.

O trabalho desenvolvido nessa tese visa o desenvolvimento de técnicas explicativas para a fácil interpretação de padrões de similaridade presentes em projeções multidimensionais e árvores de similaridade. Primeiro, propomos métodos que possibilitam a computação eficiente de árvores de similaridade para grandes conjuntos de dados, e também a sua explicação visual em multiescala, ou seja, em vários níveis de detalhe. Também propomos modos de construir representações simplificadas de árvores de similaridade, e desse modo estender ainda mais a sua escalabilidade visual. Segundo, propomos métodos para explicar visualmente projeções multidimensionais em termos de grupos de observções relacionadas, detectadas e anotadas automaticamente para explicitar aspectos de sua similaridade no espaço de alta dimensionalidade. Mostramos em seguida como esses mecanismos explicativos podem ser adaptados para lidar com dados de natureza estática e dependentes no tempo. Nossas técnicas são construídas visando fácil utilização, funcionamento semi automático, aplicação em quaisquer tipos de dados multidimensionais quantitativos e quaisquer técnicas de projeção multidimensional. Demostramos a sua utilização em uma variedade de conjuntos de dados reais, obtidos a partir de coleções de imagens, arquivos textuais, medições científicas e de engenharia de software.

Palavras-chave:
Visualização, Análise visual, Dados multidimensionais, Computação gráfica

Contents

Introduction

1

In the last years, the world has witnessed several disruptive changes created by technology. Among these, information technology (IT) stands out at the forefront of changes that we see both in daily life and also in professional and industrial activities. Within the IT context, we can further classify change as related to hardware, data, and software. *Hardware* capabilities have enormously increased in terms of computing power, computing speed, and ease of use, on the one hand, while form factors and prices continuously drop. For example, a consumer-grade Graphics Processing Unit (GPU) in a modern mobile phone provides considerably more computing power than a state-of-the-art desktop workstation fifteen years ago. *Data* is the second key element of the observed change. The size (Volume), speed of change (Velocity), and diversity of types of datasets being acquired and manipulated (Variability) have increased hundreds of times in the last years, leading to what is currently known as the 'big data revolution' with its 3V challenges. Finally, *software* algorithms, computing paradigms, and development tools have widely diversified, leading to increasingly sophisticated applications, running on all types of hardware, and addressing all types of data out there.

However, such large developments always come paired with equally large challenges. In this thesis, we focus on the pair *data* and *software*, in the following sense. First, we observe that the 3V characteristics of *data* are making the efficient and effective use of large, complex, hybrid, and time-dependent datasets increasingly harder in terms of using such datasets in concrete applications. The increase of volume does not come as a surprise: Large datasets are, by definition, harder to process (efficiently, in any case) than smaller datasets, all other things being kept equal. The same holds if we compare time-dependent (dynamic) datasets with time-independent (static) datasets. However, *variability* poses several specific challenges. Simply put, given a dataset of N data items (also called observations or sample points), it is far harder to analyze, process, or even make sense of such a dataset in the case the N points have different structure and/or semantics than in the case they represent the same (simple) type of data elements. This implies increasingly complex challenges in terms of the design of *software* to handle such datasets: Even in cases when the problem to solve on the data at hand is relatively well understood, designing such software can become hard due to the 3V aspects of the underlying data. As such, the increased size, complexity, and variability of data induces increased difficulties for designing software applications to handle such data, which in the end reflect in decreased ability of addressing the end-user tasks for applications using the respective data.

Recognizing the importance of understanding big data spaces prior to designing applications to handle such data, scientists and engineers have focused on developing methods, techniques, and tools for the exploration, analysis, and interpretation of such data spaces. Such methods have emerged from many disciplines, including data mining [67], machine learning [124], information systems [176], database technology [5], cloud computing [23], and data visualization [183]. During this development, it has been recognized that, as the 3V aspects of big data become increasingly large and intertwined, solutions for data exploration need to include aspects taken from all above-mentioned disciplines, in order to approach the challenges at hand from a multi-facet perspective.

Within this context, *visual analytics* has emerged as one of the key approaches to understanding big data. Visual analytics, a synthesis of data visualization, data mining, and human-computer interaction, focuses on the development of theories, techniques, and tools that facilitate analytical reasoning about (big) data by the use of interactive visual interfaces [207, 31]. Since its inception, roughly one decade ago, visual analytics has known an impressive development, currently addressing data-exploration problems in application domains as diverse as business intelligence, medical science, engineering sciences, chemistry, physics, and social sciences. One of the key tenets that makes visual analytics an attractive (and effective) solution to the so-called 'sensemaking' from data, is its iterative approach to the task: The classical process of defining a hypothesis concerning a phenomenon to which the data relates, defining a model to capture the hypothesis, instantiating the model, and verifying its results concerning the hypothesis, is kept in place. However, this process is both significantly accelerated and made iterative by the use of visual interactive tools. Simply put, end users can perform all above steps rapidly and intuitively; and multiple exploration paths can be spawned, explored, and either discarded or further refined, as needed, by iterating the steps of the data-exploration process as many times as needed until a conclusion is formed – or, in visual analytics terminology, *insight* is obtained or one has been able to *make sense* of the data [101, 100].

However, as we shall see, while visual analytics promises to be an effective and efficient solution to the problem of understanding data to further support the construction of end-user applications or to provide the final insights required by such use-cases, it is also confronted by several technical challenges. In this thesis, we explore one type of such challenges, related to the *dimensionality* of the data at hand. As we explain next in Sec. 1.1, datasets having high dimensionality are very hard for humans to comprehend, thereby hard to intuitively and easily manipulate in a visual analytics pipeline. To this end, several specific techniques have been designed, of which the so-called multidimensional projections have specific properties which make them of high potential, as we next explain in Sec. 1.2. The core research questions of this thesis related to the refinement of multidimensional projections for supporting the understanding of high-dimensional data are next distilled in Sec. 1.3. This introductory chapter is closed by the presentation of an outline of this thesis in Sec. 1.4.

1.1 Multidimensional data

Recalling the 3V aspects of big data, let us focus on the Volume, or size, aspect. Roughly speaking, most data collections, or *datasets*, are structured as a set of so-called observations, also called data points, measurements, records, or samples. Each such observation is a record of the same type of aspect of the underlying phenomenon being studied. Therefore, observations have the same structure, defined in terms of *what* they measure, but take different values, defined in terms of what the *values* of the measurements are. Per observation, one can measure a single or multiple data values. When such multiple measurements are taken per observation, one speaks of a *multidimensional* dataset. Here, each dimension, also called attribute, variable, or feature, describes a different aspect of the data at hand. Typically, all observations have values for the same set of dimensions. As such, a multidimensional dataset can be thought of as a (large) table having one row per observation and one column per dimension.
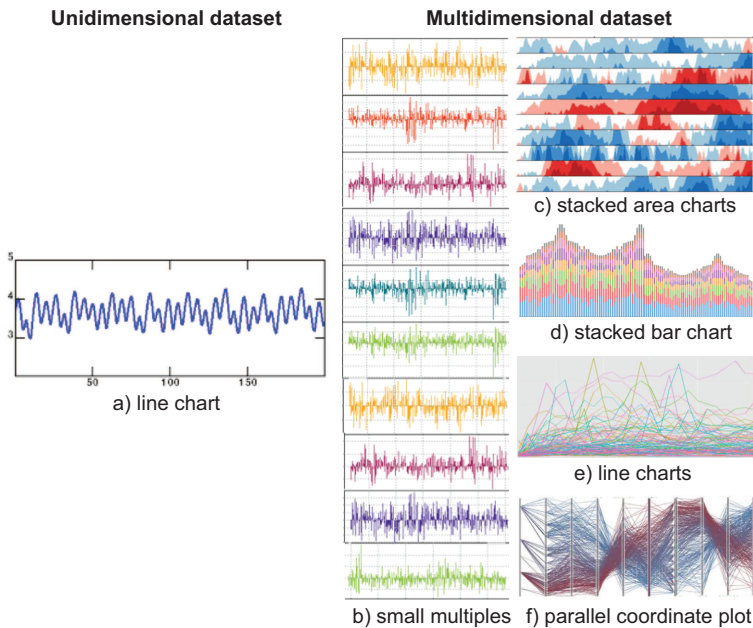
Figure 1.2: Left: Unidimensional dataset visualized with a simple line chart (a). Right: Multidimensional dataset visualized with (b) small-multiple line charts; (c) stacked area charts; (d) stacked bar charts; (e) line charts; (f) parallel coordinates.

Multidimensional datasets are common in virtually all domains of science and engineering where (big) data is collected, analyzed, and visually explored. For example, multimedia data (sound, image, and video) collections can be thought of a set of observations, each being an individual content item. Here, dimensions are typically extracted using machine learning techniques to characterize

the observations in terms of their similarity, as this is perceived in the application domain [74]. This enables next applications such as content-based retrieval and browsing of large multimedia collections [95]. Collections of text documents can also be thought of as multidimensional datasets. Here, an observation is a document, and its dimensions are the frequencies of specific terms (keywords) computed over the entire document collection [34]. Typical applications based on this model are document retrieval, document indexing, and text collection summarizations [141]. Many other application domains use multidimensional datasets, such as business intelligence [70, 18], weather forecast [32], software maintenance [106, 153], and customer analysis [33]. Several of these application domains will be detailed in the context of the techniques and applications discussed later on in this thesis.

Given the above model, it is clear that the size of a dataset is given by the size, or number of cells, of the data table that would contain values for all observations over all dimensions. While this simple model captures the size of a dataset quite well, it does not explain the various difficulties encountered when analyzing multidimensional data. To understand this, consider a simple unidimensional dataset having one attribute measured over 1000 observations, such as a time series depicting the evolution of a stock price. The size of this dataset is, obviously, 1000 numerical data values. Consider now a 10-dimensional dataset having ten attributes measured over 100 observations, such as the description of 100 types of computers along attributes such as price, CPU speed, memory size, hard disk size, graphics card speed, and weight, up to a total of 10 such attributes. The sizes of the two datasets are identical. However, the unidimensional dataset can be visualized very easily, *e.g.* by using a classical line chart (Fig. 1.2a). This visualization is very simple to interpret – most end users would be able to use it to detect patterns such as peaks, valleys, plateaus, instability regions, and sustained growth or decline. Moreover, line charts are very scalable to the number of observations, and simple and fast to compute. In contrast, visualizing the second 10-dimensional dataset is far more challenging. Using simple charts such as one line chart per dimension (an instance of a broader class of techniques known as 'small multiples', Fig. 1.2b), stacked area charts (Fig. 1.2c), stacked bar charts (Fig. 1.2d), or superimposed line charts (Fig. 1.2e) do not reveal most of the interesting patterns in the dataset, such as outliers, correlations of dimensions, groups of similar observations, or independent dimensions.

1.2   The need for multidimensional data visualization

The previous section has outlined some of the challenges involving the understanding of data patterns existing in high-dimensional datasets. This problem has received attention in several research fields, of which two most important ones are *machine learning* and *data visualization*, discussed below.

1.2.1 Machine learning

In machine learning, multidimensional data is essentially analyzed by using various (semi)automatic methods that search for patterns of interest in a given dataset [124]. Such patterns can be as simple as finding correlated or independent dimensions, finding groups of similar observations, finding outliers [79], ranging up to more complex patterns such as shapes and distributions [15]. The key advantage of such approaches is *automation*: Given that one knows which are the patterns of interest, and how to describe them in terms of combinations of a dataset's dimension, one can design algorithms to automatically search for such patterns in the data, and report their presence to the end user [74]. This process can function largely without any human intervention, such as in the case of classifiers and clustering techniques being used to group large data collections into types, or classes, that reflect application-specific concerns [92, 107]. However, this advantage of automated methods does not come for free – one must know in advance how to design, train, and fine-tune machine learning algorithms to detect the desired patterns of interest. In many cases, doing this is far from simple. Challenges that appear during this process include the following:

- *Description:* It is far from obvious how to describe such patterns in terms of the available data dimensions. For example, consider the task of capturing the similarity of images in a collection for the purpose of classification or pattern recognition. To this end, a multitude of features can be extracted from images [46]. From these, how to find those features that are the most effective in capturing specific patterns present in the images?

- *Decision support:* Consider the construction of a classifier tool used to classify, or label, observations – a standard task in machine learning [47]. In supervised learning, this is typically done by training the classifier using a limited-size set of manually labeled examples. However, it is far from clear how to optimally select such a set, or how to fine-tune the various parameters of the machine learning technique being used so it achieves the desired classification performance next. To do this, the designer would ideally need to understand how the data is organized in the high-dimensional space.

- *Discovering the unknown:* There are also cases when one does not know what to look for specifically in the data, *i.e.*, one is interested to 'discover the unknown' insights that may be part of the data at hand [183]. This task cannot be approached by standard machine learning techniques, since we do not know what to search for in the first place.

1.2.2 Multidimensional visualization

The above challenges of machine learning are addressed by a second class of methods – multidimensional data visualization. Multidimensional data visualiza-

tion is a subfield of information visualization, the branch of data visualization
that focuses on the visual depiction of large, non-spatial, abstract data spaces
[126, 172]. Like classical scientific visualization, multidimensional visualization
attempts to map the available data to visual variables (position, shape, size, ori-
entation, texture, color, animation) so that patterns of interest, as well as unex-
pected patterns, are easily detectable by looking at the produced images [114].
Many multidimensional data visualization techniques have been proposed, start-
ing from simple (but usually limited) ones, such as the charts shown in Fig. 1.2a-e,
to more advanced ones, such as the parallel coordinate plots (Fig. 1.2f) [114]. A
comprehensive survey of these techniques will be presented later in Chapter 2,
Sec. 2.3.1.

   In the following, we focus on two classes of multidimensional visualization tech-
niques – projections and relational data visualizations. As we shall see, these are
related in intimate ways, and these relationships will lead us to the formulation
of our research questions, described next in Sec. 1.3.
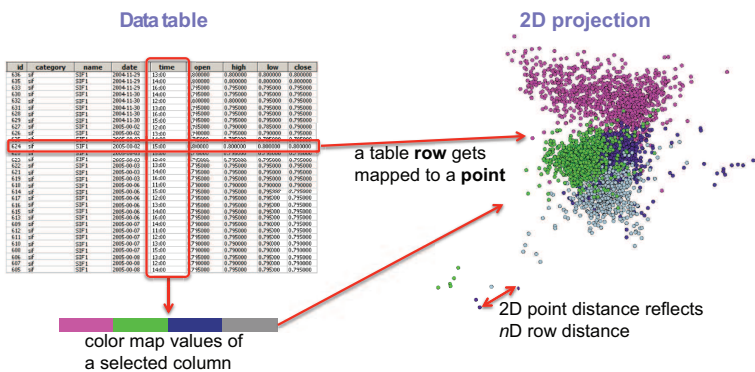
Figure 1.3: Conceptual operation of a projection. Image taken from [38].

1.2.2.1 *Dimensionality reduction techniques*

Within the large palette of multidimensional visualization techniques, *projections*
or *dimensionality reduction* techniques occupy a particular place. Simply put, pro-
jections address one of the key difficulties of understanding multidimensional
data – the large number of dimensions – by reducing this number while main-
taining important interrelations between data points. Given a high-dimensional
input dataset, a projection technique outputs a low-dimensional dataset, typically
having the same number of observations (Fig. 1.3). The dimensions of the latter
usually aggregate or amalgamate the data variation present along the original di-
mensions of the input dataset in ways that preserve the so-called *data structure*
– that is, groups of similar observations, outliers, trends, and correlations. When
the number of dimensions of the output dataset is two or three, such datasets can

be directly visualized using, for example, 2D and 3D scatterplots [171]. If the dimensionality reduction was properly done, in the sense of preservation of the data structure, the users should be able to employ such scatterplots to reason about the high-dimensional data patterns. For instance, if the projection preserves distances between observations or neighbors of observations, then we can see whether the input data consists of coherent groups of highly similar observations by looking for groups of densely-packed points in the projection. As we shall see in Chapter 2, Sec.2.4, many multidimensional projection techniques exist that cover the entire spectrum from fast and simple to implement, but less able to preserve data structure [96], to sophisticated techniques that can be tuned to preserve various aspects of the data and that can handle datasets of hundreds of thousands of observations and hundreds of dimensions [95].

Projections combine several attractive aspects, as compared to other visualization techniques for multidimensional data: Like certain machine learning techniques, they can be applied with minimal or no user intervention; they are arguably the most visually scalable visualization techniques out there for high-dimensional data – an observation having tens up to hundreds of dimensions is reduced to a single scatterplot point; recent projection techniques are accurate, robust, and can handle large volumes of data (both in number of observations and number of dimensions), and data of different attribute types, efficiently; and understanding the metaphor of a scatterplot is arguably simpler for typical end users than other multidimensional visualization metaphors such as parallel coordinates.

However, projections also have a significant drawback. As mentioned earlier in this section, dimensionality reduction typically takes the form of synthesizing a low number of dimensions from the original data dimensions [164]. While they enable the direct creation of 2D or 3D scatterplots, these synthetic dimensions do not have a *direct* meaning to the end users. As such, interpreting the patterns present in the projection, or performing the so-called 'inverse mapping' from the visualization to the original data [183] is very hard. In contrast, the vast majority, if not all, of the other multidimensional visualization methods explicitly encode the original data dimensions in the final image. This makes projections much more challenging to use. For example: Consider the 2D projection shown in Fig. 1.3 right. In here, we see a large central cluster of close points, surrounded by several outliers. In turn, the central cluster appears to be divided into an upper sub-cluster and the remaining points. If we assume that the projection being used to create this image preserves distances between points (as is the case with many of the existing multidimensional projection techniques), then it means that there are several distinct groups of highly similar points in the data (the aforementioned clusters), and a few points which are very different from all the others (the aforementioned outliers). The question is: Which dimensions, and which dimension values, can explain these patterns? Without knowing these, the projection tells us that there is *some* structure in the data, but not *what* that structure means.
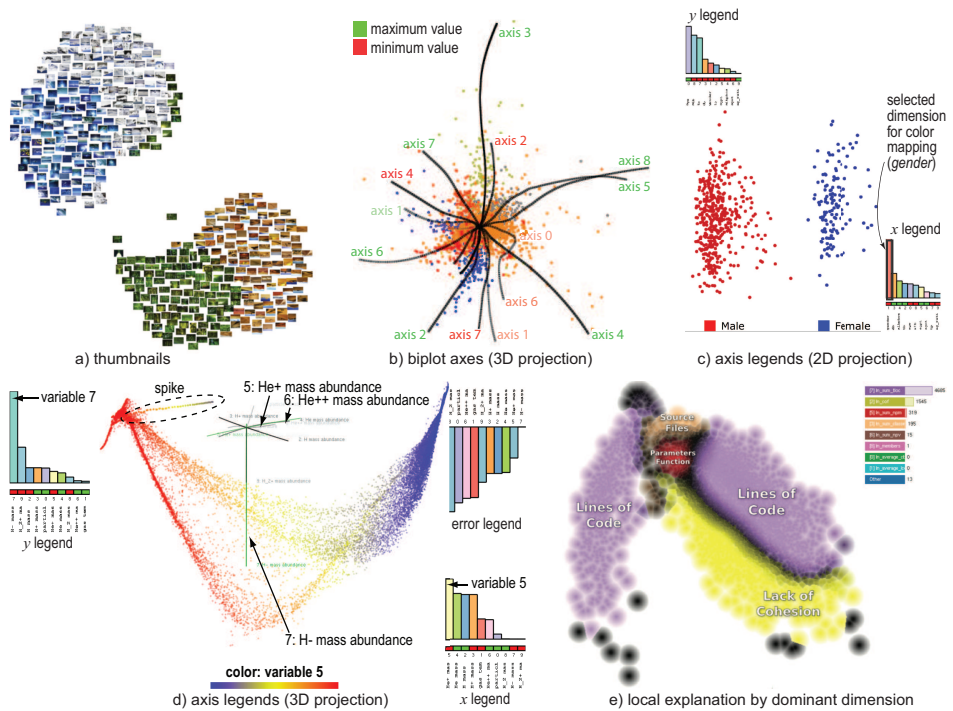
Figure 1.4: Explanatory techniques for projections. (a) Thumbnails. (b) Biplot axes. (c) Axis legends and color coding for 2D projection. (d) Axis legends and color coding for 3D projection. (e) Local explanation proposed in this thesis, Chapter 5. Image taken from [38].

The above issue is addressed by so-called *explanatory techniques* for multidimensional projections. Simply put, such techniques annotate, or enrich, a raw projection scatterplot with visual cues that explain the patterns present in it. Among other aspects, such cues aim to 'put back' summaries of the original high-dimensional information, such as dimension names and values, into the projected data. In visualization terms, they act as *legends* that allow the user to interpret, or explain, the projection [183]. Several such techniques have been proposed in the literature. The simplest explanatory techniques involve color-coding the projected points by the values of a user-chosen dimension, as done in the example in Fig. 1.3; and interactively brushing the points to show details on demand [211]. More complex techniques exist as well. For instance, one can cluster the projected points and annotate the resulting point groups with dimension names [143]. Biplot axes can be drawn to indicate the directions of maximal variation of the original dimensions [71, 28], see also Fig. 1.4b. Thumbnail-like icons can be drawn atop the projected point positions in case one can effectively summarize the high-dimensional data points by such icons, such as when the observations are actual images (Fig. 1.4a). Legends can be used to explain the screen's horizontal and vertical axes in terms of the original dimensions [18, 28] (Fig. 1.4c,d). However useful, such techniques still have several drawbacks. First, except the local brushing and the clustering methods, all other methods are *global* by nature, *i.e.*, explain an entire projection rather than the patterns it contains. Separately, local brushing and clustering require a non-negligible amount of input from the end user, and thus are not optimal in terms of usability. As we shall see in Chapters 5 and 6, more intuitive explanatory techniques for projections can be imagined, such as the local color-coding and labeling illustrated in Fig. 1.4e.

1.2.2.2 *Relational data visualization*

Projections, as a visualization technique for multidimensional data, are closely related to *relational* data visualization methods, in several ways. First, one can consider the (square) distance matrix containing the distances, in the original data space, among all observations. Lower thresholding this matrix by some preset distance value yields a sparse matrix that encodes the distances between the highly similar observations in the dataset. In turn, this matrix induces a graph relating these observations. Computing a projection that preserves well the (small) distances is roughly equivalent to computing a layout of the (sparse) graph where connected nodes are placed close to each other. This relationship has been used both to compute layouts of graphs by first embedding the graphs in a high-dimensional space and next project this space to 2D using, for instance, principal component analysis (PCA) [77]; and also by computing layouts of multivariate graphs by computing a 2D projection of a distance matrix that captures both the similarity of points in terms of data attributes and their graph connections [119].

A second relation between projections and relational data visualization can be found if we consider the errors produced by a projection. Understanding these
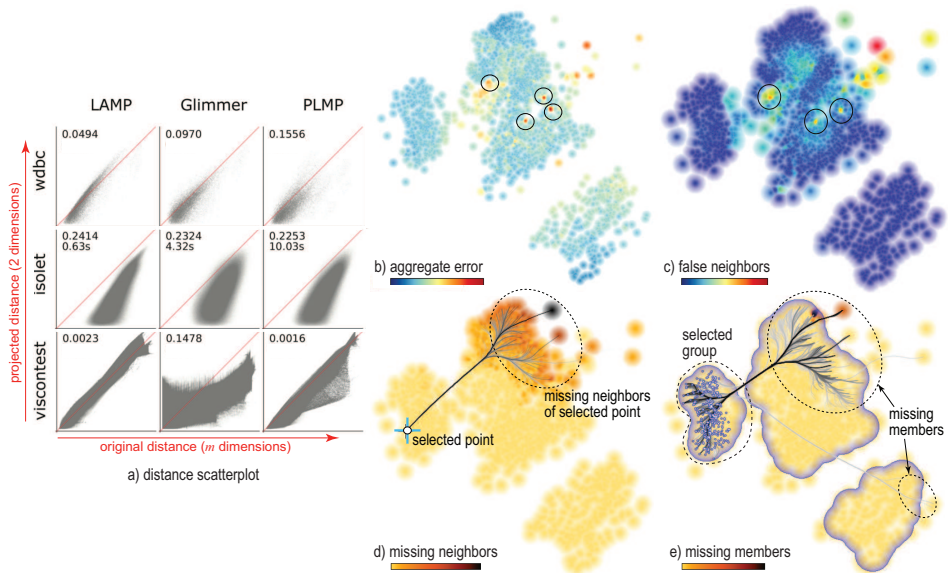
Figure 1.5: Visualization of projection errors. (a) Distance scatterplot matrix. (b) Aggregate projection error. (c) False neighbors of projected points. (d) Missing neighbors of a single selected point. (e) Missing members of a selected point group. Image taken from [38].

errors is very important for the usefulness of a projection – we can use the projection as a 'proxy' to reason about the high-dimensional data patterns only if the projection accurately captures these patterns. Defining (and next understanding) projection errors is by excellence a task that involves *relations* between points. Figure 1.5 illustrates this. In Fig. 1.5a, we show a scatterplot matrix (SPLOM) where each cell displays the scatterplot of the high-dimensional distances between point-pairs *vs* the distances between the same point-pairs in the projection [95]. A good projection should yield scatterplots close to the matrix cell diagonals. Figure 1.5b shows a 2D projection whose points x_i are colored by the sum of distance errors between x_i and all other projected points $x_j \neq x_i$ [118]. Figure 1.5c shows a similar image to Fig. 1.5b, where points x_i are colored by the number of so-called 'false neighbors', *i.e.*, points which are placed in 2D closer than they are to x_i in the original data space [120]. As in image (a), images (b) and (c) consider all point-pair relations when computing errors. In contrast, Fig. 1.5d shows the so-called 'missing neighbors' of a selected point x_i, *i.e.*, the points which are projected too far away from x_i than they actually are in the high-dimensional data [118]. Finally, Fig. 1.5e shows the so-called 'missing group members', *i.e.*, points which are projected too far away from a selected compact group of points than they actually are in the original dataset. In images (d) and (e), thus, the relations of one, respectively a subset, of points with all other points are visualized. The

relational nature of such data is made explicit by the use of edge bundling [213] to show the missing neighbors and missing group members in Figs. 1.5d,e.

Summarizing the above, we see that a key element for both computing and interpreting projections regards distances between observations – indeed, the majority of patterns that create the so-called data structure mentioned previously are constructed by considering the position of an observation *in relation to* other observations. Separately, we see that such relations involve reasoning about the distance between, or similarity of, observations. Hence, *similarity* of observations is an essential element for understanding projections. Separately, we also see that existing techniques can show which points are similar in a projection, but not *why* these points are similar. This complements our earlier observation in Sec. 1.2.2.1 that additional explanatory mechanisms for multidimensional projections are required.

This insight concerning the importance of understanding similarity can be lifted to the exploration of multidimensional datasets with techniques beyond projections. Consider the construction of a hierarchy of observations in terms of their similarity, as if one applied a bottom-up agglomerative clustering process [93]. When displayed, this hierarchy, also called a *similarity tree*, shows groups of similar points at multiple scales, and without having to choose an explicit scale like in standard cluster labeling [136]. By using a hierarchy to explicitly display data similarity, trees have also the advantage of avoiding the ambiguity often observed on multidimensional projections, which can easily lead to wrong conclusions about the dataset. Moreover, by using suitable graph layout techniques, similarity trees can be drawn by avoiding various clutter issues that appear when displaying large multidimensional datasets using projections. In all the examples mentioned above, relational visualization techniques share similarities with projections in terms of creating the same type of visual 'backbone' (points representing observations, placed in the 2D space so as to reflect data similarities), and also in terms of high visual scalability in both the observation and dimension counts. However, they also share some common problems, most notably in terms of the ease for understanding what groups of related observations mean. A separate problem of similarity trees is that they scale poorly with the dataset size, making them unsuitable for datasets containing hundreds of thousands of observations or more.

1.3   Research questions

Summarizing the above introduction on multidimensional data visualization, we see that *similarity* is a key concept to understanding multidimensional data patterns, whether these are groups of related points or projection errors (in a projection) or groups of related points (in a similarity tree). Both projections and similarity trees use proximity in the 2D space to depict similarity in the original high-dimensional space, whereby one can easily detect patterns such as groups of related observations and isolated outliers. However, both projections and sim-

ilarity trees are limited in explaining what such patterns actually mean in terms of the original data dimensions. Several explanatory techniques for such visual metaphors exist, but they either demand a non-negligible amount of effort from the end user or provide explanations which, in our view, are not always intuitive.

Given the above, we can now formulate the central research question of this thesis:

How can we help a wide range of users to easily visually reason about, and explore, the notion of similarity present in large multidimensional datasets?

Given that we focus on projections and similarity trees as visualization metaphors, our aim is thus to enrich these techniques with explanatory mechanisms that keep their various advantages but decrease their disadvantages. As such, we can refine the above research question into two sub-questions:

RQ1: How can we enhance similarity trees so they become computationally scalable and also provide local and automatic explanations of their subtrees?

RQ2: How can we enhance multidimensional projections with explanatory mechanisms that provide local and automatic explanations of the perceived patterns?

1.4   Structure of this thesis

The structure of this thesis follows the subdivision of its main research question into sub-questions (see Sec. 1.3), as outlined next.

In Chapter 2, we review related work covering the main domains of research related to our own research questions. We start by discussing the structure and organization of multidimensional data, which underlies all our work. Next, we discuss clustering techniques, as these techniques are instrumental to the extraction of patterns in terms of groups of similar entities, and are also a key ingredient of the construction of similarity trees, and of the multilevel presentation of large datasets more generally. Next, we discuss visualization techniques for multidimensional data, with a particular focus on projections and hierarchical (tree) structures.

In Chapter 3, we propose our improvements to the construction and visual exploration of multidimensional datasets using similarity trees. The first proposed improvement here addresses the computational scalability of similarity trees in terms of a large number of instances. The second class of proposed improvements concerns the explanation of a computed similarity tree, which is done in terms of annotating the resulting tree-based visualization with colors, textures, and labels so as to depict summarizations of the data subsets encoded by the various tree parts.

Chapter 4 extends the visual scalability of the proposed similarity tree design described in Chapter 3. Specifically, we show how we can adapt and extend hierarchical edge bundling so as to construct compact summarizations of large similarity trees. Separately, we show how our proposal extends the flexibility of edge bundling as a tool for simplifying the depiction of large graphs, by adding semantic information obtained from the underlying multidimensional data. Together, Chapters 3 and 4 thus address our first research question **RQ1**.

Chapter 5 switches our focus to projections. We present here a novel technique for the visual explanation of multidimensional projections in terms of groups of similar points. In contrast to the construction of a hierarchy based on the data similarity in the original high-dimensional data space, we now exploit similarity in the projection space. As such, the resulting explanatory visualization focuses on explaining patterns present in the projection, rather than patterns present in the original dataset.

Chapter 6 extends our explanatory techniques for multidimensional projections from the static (time-independent) case discussed in Chapter 5 to treat dynamic, or time-dependent, multidimensional datasets. For this, we use a projection technique able to handle time-dependent datasets, and also leverage the use of edge bundling to depict the space-time aspects of the underlying multidimensional data. Together, Chapters 5 and 6 thus address our second research question **RQ2**.

Chapter 7 concludes the thesis by discussing the various strengths and weaknesses of the proposed techniques, their interdependencies, and also by outlining interesting directions for future work in the area of visually exploring multidimensional datasets.

Related Work

As outlined in Chapter 1, this thesis addresses the goal of exploring similarity in multidimensional datasets by means of projections and similarity trees visualization metaphors. To provide background for this research, we survey in this chapter the main techniques related to our field of research, structured as follows. In Section 2.1, we introduce the main concepts and notations related to multidimensional data. In Section 2.2, we overview data clustering and aggregation, an important aid for the handling of large datasets when producing visualizations thereof. Section 2.3 overviews the main techniques known in the literature for the visualization of multidimensional data, thereby laying the context in which we will work next. Here, we dedicate specific attention to the visualization techniques that we will reuse, adapt, or extend, namely projections and visualization of relationships. Section 2.4 overviews techniques for computing multidimensional projections, as well as for visually explaining such projections.

2.1 Introductory Concepts

2.1.1 Multidimensional datasets

Multidimensional datasets are collections of data entities, also called data points, observations, measurements, samples, or records. Informally put, each observation describes the measurement of several properties of a given phenomenon. Typically, all observations share the same number and type of measured properties. These properties are also called attributes, dimensions, variables, or features. Informally put, multidimensional datasets can be thus modeled as data tables, where each table row describes one observation, and each column describes one dimension. More formally put, let n be the number of data points and m the number of dimensions respectively; and let $\mathbb{D}_1 \dots \mathbb{D}_m$ be the domains of the respective m dimensions. Hence, an observation $\mathbf{x}_i$ is a point of the space $\mathbb{D}_1 \times \mathbb{D}_2 \dots \times \mathbb{D}_m$. We denote the set of all observations under consideration, thus a multidimensional dataset, by $D = \{\mathbf{x}_i\}$. Each observation is thus a tuple $\mathbf{x}_i = (x_i^1, \dots, x_i^m)$, where $x_i^j \in \mathbb{D}_j$, $1 \leqslant j \leqslant m$ denotes the value of its jth dimension. Finally, we denote the values of dimension j over all n observations by the vector $\mathbf{x}^j = (x_1^j, \dots, x_n^j) \subset \mathbb{D}_j^n$.

Multidimensional datasets can be obtained from various sources. Without being formal, we classify these into two categories. First, multidimensional datasets can be the actual form in which data is stored in an application domain. For example, in an Electronic Patient Record (EPR) database, patients can be seen as observa-

tions. Each patient record has several dimensions, such as the name of the patient, age, medication, duration of treatment, condition, and so on. More generally, any application domain where data is stored as a table where rows encode individual observations, can be thought of as generating multidimensional datasets. When data is stored into multiple tables, linked *e.g.* by means of foreign keys, 'simple' data tables that conform to our above mentioned model can be generated on-the-fly using joins. As such, the table metaphor is both an intuitive and practical way to think of multidimensional data. This is also seen in the advent of specialized information visualization software that revolves around the table concept [179].

Apart from the above direct interpretation of existing data sources as multidimensional datasets, such datasets can be also *derived* from existing data by a process commonly known as *feature extraction*. Here, the original data usually comes in a form where separate dimensions are either not present, or not suitable for further analysis [74]. A simple example hereof are collections of images. Typical tasks involving such data revolve around finding items similar to a given data item (query-by-example, or content-based retrieval [115, 42]), grouping data items in subsets of similar items, or classifying items into several categories [104]. For all these tasks, one needs to extract relevant dimensions from the data items in order to support *comparison*, the key operation that next enables computing similarity, which next enables the computation of groups and outliers. For this, several techniques collectively known under the name *feature extraction* are used. For our example of image data, such features involve usually histograms of color components, edges, and textures [130, 95]. For other datasets, such as audio data, one can use histograms describing the signal's frequency spectrum [123, 29]. For text data, frequencies of the most common terms found in a document collection are typically used [1]. In all these cases, the output of the feature extraction step is a multidimensional dataset, where each dimension captures the values of one of the computed features. As such, for our scope, multidimensional datasets will be treated identically, whether obtained directly from a given application domain, or by feature extraction.

A further difference between multidimensional datasets regards the types of the domains of definition $\mathbb{D}_i$ of their dimensions. Following information visualization terminology, these domains can be characterized based on the operations that their elements, *i.e.*, data attributes, admit. The following main types of attributes are recognized [126, 183]:

- *categorical:* Categorical attributes admit only exact comparison, *i.e.*, the operators $=$ and $\neq$. They are defined over countable sets. They usually describe types or classes, such as, for instance, gender, label, or product brand type;

- *ordinal:* Ordinal attributes add the comparison operators $<$ and $>$ to categorical attributes. They are defined over countable ordered sets. They describe ranks, *e.g.* scores on a Likert scale;

- *integral:* Integral attributes essentially model values over (subsets of) $\mathbb{N}$ or $\mathbb{Z}$. They add the difference, or distance, operator $\| \cdot \|$ and the sum operator $+$ to ordinal attributes. They usually describe counts, such as number of persons in a census;

- *quantitative:* Quantitative, sometimes also called continuous, attributes model (subsets of) $\mathbb{R}$. They add the multiplication with a real weight to integral attributes. They describe values over a (subset) of an uncountable set, such as $\mathbb{R}$. Also, when they describe dependent dimensions, the dependency is usually implied by a (Cauchy or Lipschitz) continuous function. They are the most 'powerful' type of attributes.

Apart from the above basic attribute types, many other domain-dependent attribute types can be defined. For instance, one can consider images, text, or source code to be valid domains $\mathbb{D}_i$. However, since such domains do not generally admit a simple-to-use algebra providing operations, they are in practice reduced to the above-mentioned basic domains by means of feature extraction. Also, relations, such as edges in a graph or tree, can be seen as a separate attribute type [183]. However, a relation is defined by definition by at least *two* observations, whereas all above attribute types can be defined by a single observation. This sets relations apart from all other attribute types discussed here. Also, different terminologies exist for the same concepts. For example, the well-known Tableau feamework for visual analytics calls quantitative and integral attributes *measures* and ordinal and categorical attributes *dimensions* [179], and adds a further distinction into discrete attributes (all except the quantitative ones) and continuous attributes (what we call here quantitative attributes). For clarity, we use the terminology presented above in our text in the remainder of this thesis.

Given the above basic attribute types, a multidimensional dataset can be characterized by the types of the attributes of its observations. Two main classes of datasets exist in this sense:

- *uniform* datasets: These are datasets where all domains $\mathbb{D}_i$ are of the same type $\mathbb{D}$. In this case, a dataset having m dimensions basically has observations taking values in $\mathbb{D}^m$. Uniform datasets are relatively simple to handle once we have a well-defined set of properties for the domain $\mathbb{D}$;

- *hybrid* datasets: Also called nonuniform or mixed datasets, the domains $\mathbb{D}_i$ of the individual m dimensions have different types. For example, a client record dataset where, per patient, one stores a name, birth date, and billing amount is a three-dimensional dataset of attributes of type text, integral, and quantitative. Hybrid datasets are (considerably) more complex to handle than uniform datasets due to the diversity of operations that the individual attributes support. In practice, a simple way to approach this is to reduce hybrid datasets to uniform categorical ones, *e.g.* by binning the quantitative, ordinal, and integral attributes [18].

Covering all combinations of attribute types in a visualization work is a tremendous endeavor, and also one that would distract the reader from the main goal of the thesis. As such, we focus on uniform quantitative datasets in the remainder of this work, as these are, arguably, a generalization of the other attribute types. When our concrete datasets will have different (mixed) types, we will show how uniform quantitative datasets can be derived from those respective types.

2.1.2 Distance Metrics

One important step in data analysis is to define a meaningful way to compare data observations. In particular, this step is essential to our thesis work, where we want to be able to depict similarities of data items. For comparison, several distance metrics are used. A distance metric $\delta : \mathbb{D}^m \times \mathbb{D}^m \to \mathbb{R}^+, \delta(\mathbf{x}_i, \mathbf{x}_j)$, takes two observations $\mathbf{x}_i, \mathbf{x}_j$ from some dataset $X \subset \mathbb{D}^m$ as arguments and produces a positive real number as output. A true metric has to satisfy the following conditions:

- **Positivity:** $\delta(\mathbf{x}_i, \mathbf{x}_j) \geqslant 0, \forall i, j;$

- **Symmetry:** $\delta(\mathbf{x}_i, \mathbf{x}_j) = \delta(\mathbf{x}_j, \mathbf{x}_i), \forall i, j;$

- **Reflexivity:** $\delta(\mathbf{x}_i, \mathbf{x}_i) = 0, \forall i, j;$

- **Triangle inequality:** $\delta(\mathbf{x}_i, \mathbf{x}_j) \leqslant \delta(\mathbf{x}_i, \mathbf{x}_k) + \delta(\mathbf{x}_k, \mathbf{x}_j), \forall i, j, k.$

The first three conditions are easy to understand, as they directly map to the generally accepted concept of dissimilarity or distance between any items. The triangle inequality intuitively states that the 'direct path' between two observations is always equal or shorter than a path connecting the same two observations via a non-straight path. In other words, a distance between a pair of observations cannot be shortened by trying to use a third observation as 'shortcut'.

If δ fulfills the following additional restriction,

- $\delta(\mathbf{x}_i, \mathbf{x}_j) \leqslant \max\left(\delta(\mathbf{x}_i, \mathbf{x}_k), \delta(\mathbf{x}_j, \mathbf{x}_k)\right), \forall i, j, k$

it is called ultrametric. If δ is ultrametric, given any combination of three observations $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k$, the distances $\delta \mathbf{x}_i, \mathbf{x}_k$ and $\delta \mathbf{x}_j, \mathbf{x}_k$, will 'contain the distance $'delta \mathbf{x}_i, \mathbf{x}_j$ [85].

Many concrete distance metrics exist that satisfy the above properties. We outline below the most common such metrics in the context of analyzing and visualizing multidimensional data.

Minkowski metric: The Minkowski, or L_p, norm is a family of distance metrics defined by

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{l=1}^{n} \left| x_i^l - x_j^l \right|^p \right)^{1/p}. \tag{2.1}$$

For $p = 1$, the Minkowski metric becomes the *Manhattan distance*, also kown as *city-block* distance. This distance is computed as if an axis-aligned grid is positioned over the data space, which allows distances to be computed only along paths following the grid. For $p = 2$, the Minkowski metric becomes the *Euclidean distance*, which can be intuitively seen as the length of a 'straight" line segment between two observations. For $p = \infty$, the Minkowski distance becomes the *Chebychev distance*, *i.e.*, $\delta(\mathbf{x}_i, \mathbf{x}_j) = \max_{1 \leqslant l \leqslant n} \left| \mathbf{x}_i^l - \mathbf{x}_j^l \right|$, which considers that only the dimension yielding the largest distance matters.

Mahalanobis metric: This metric takes into account the data distribution. It uses the inverse of the dataset's covariance matrix to weigh each dimension in the distance computation. It is define by

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i - \mathbf{x}_j\right)^\mathsf{T} Cov(X)^{-1} \left(\mathbf{x}_i - \mathbf{x}_j\right) \tag{2.2}$$

where $Cov(X)$ is the covariance matrix of the dataset X and the superscript $^\mathsf{T}$ denotes transposition. It can be seen as a data transformation prior the computation of the Euclidean distance for the transformed data. Intuitively, this metric is similar to the Minkowski metric, but adapts itself to normalize the data dimensions as a function of the distribution of attribute values over the dataset X. One of its major drawbacks is the cost on computing the inverse of the covariance matrix.

Cosine metric: Intuitively, this distance measures the angle of two multidimensional vectors, not considering their magnitudes. As such, this metric is useful in cases where observations which are scaled versions of each other are considered to be identical. It is defined by

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^\mathsf{T} \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \, \|\mathbf{x}_j\|} \tag{2.3}$$

where $\|\cdot\|$ denotes the Euclidean norm and $\cdot$ denotes scalar product, respectively. This metric is frequently used in information retrieval, where the observations are sparse multidimensional vectors.

In the remainder of this thesis, we will use the various distance metrics defined here to define similarity of multidimensional items, based on the most appropriate choices implied by the nature of similarity in the studied application domains.

2.2   Clustering Techniques

Clustering, or aggregation, techniques are of fundamental importance for effective analysis of large datasets. The goal of a clustering technique is to partition the dataset into groups of similar observations, according to some distance or similarity metric, thus offering a coarser level of similarity abstraction for the

dataset [92]. In other words, given a dataset $X \subset \mathbb{D}^m$ and a distance metric $\delta : X \times X \to \mathbb{R}^+$ (Sec. 2.1.2), clustering essentially provides a 'coarsening' of δ to a function $\delta' : X' \times X' \to \mathbb{R}^+$ on a dataset $X' = \{x_i'\}$ which is a so-called partition of X, *i.e.*, $\bigcup_i x_i' = X$ and $x_i' \cap x_j' = \varnothing, \forall i \neq j$. By coarsening, we mean here that $\delta(x_i, x_j) \simeq \delta'(x_i', x_j')$ for any $x_i \in x_i'$ and $x_j \in x_j'$ and for any $x_i' \in X'$ and $x_j' \in X'$. The key idea is that this coarsening reduces the cardinality of the input dataset while keeping its data structure, *i.e.*, $|X'| < |X|$, and thus analyzing X' using δ' is less costly, and easier, than analyzing X using δ. When several versions X' of decreasing sizes $|X'|$ can be obtained by clustering X, one speaks about a multiscale clustering, or multiscale simplification, of X.

A wide spectrum of clustering techniques exist. For instance, k-ary approaches divide the dataset X into k disjoint groups; hierarchical approaches create an entire set of simplifications X_i' for various degrees of simplification, or levels of detail, *i.e.*, $|X_i'| < |X_{i+1}'|, \forall i$; fuzzy methods assign a degree of membership for each observation to every cluster, *i.e.*, do not create a strict partitioning of X, or in other words $\exists i, j, x_i' \cap x_j' \neq \varnothing$ [72].

We will next introduce several examples of clustering techniques which we considered in our research. For a complete overview including (many) other methods, we refer to [92].

K-Means: K-Means is an example of k-ary clustering technique. Here, an observation can belong to only a single cluster. The initial step of K-Means is the definition of the desired number k of clusters, which is assumed to be known beforehand. Next, representative values (centroids) for each cluster are assigned, based on the values of randomly selected observations of the dataset. Next, an iterative procedure starts to create clusters for all observations, also updating their centroids. Observations are assigned to a cluster by comparing their values to cluster representatives. An observation is assigned to the cluster for which the centroid-to-observation distance is minimal, in a greedy approach. After each observation is assigned to a cluster, the centroids are updated by taking the average value of all observations in that cluster. The procedure stops when the assignment step produces no further changes. K-Means is simple to implement and understand, but works well only for data distributions that form convex clusters and where the number k of clusters is known in advance.

DBScan: This technique belongs to the so-called density-based clustering methods which essentially define a continuous density function from the discrete samples $x_i \in X$ [55]. Methods in this category considers cluster internal similarity and a minimal density value when defining a cluster. This way, smaller groups of points, below a specified minimal density, are not included in any cluster. The algorithm starts by the definition of a distance radius ϵ, and ρ_{min}, the minimal density of a cluster. Next a random observation $x_i \in X$ is selected. The iterative procedure starts by checking how many neighbors of x_i lie within the distance ϵ. If less neighbors than ρ_{min} are found, they are marked as noise. Otherwise x_i and its

neighbors create a cluster, and this procedure is repeated to process more neighbors until a densely connected cluster is created, *i.e.*, until no more elements can be added to this cluster. Next this procedure is repeated to process another random unvisited (unassigned) point, until all points are marked as visited (assigned to a cluster). DBScan can treat convex and concave clusters of variable local point density (inter-point distance) better than K-Means. However, the setting of the parameters ϵ and ρ_{min} is not always obvious.

Hierarchical: Hierarchical clustering techniques create aggregations of data by imposing a hierarchy of closely-related observations. Essentially, the idea is to define clusters in a divide-and-conquer way, by either splitting the given dataset X recursively in smaller, and increasingly more similar, bits (divisive or top-down clustering), or by recursively aggregating elements in the dataset X in larger, and increasingly less similar, chunks (agglomerative or bottom-up clustering). The agglomerative strategy starts by defining each observation as a cluster, and next iteratively joining the most similar pairs of clusters, until only one cluster, equal to X, is left. The divisive strategy does the opposite, and starts by considering the entire dataset X as a single cluster, to next divide it in smaller partitions until all clusters have only one element. Hierarchical clustering is generic, and simple to implement, but has in general high costs ($O(N^3)$ for N observations in X). Also, once the clustering is ready, one typically obtains a binary tree having the observations $\mathbf{x_i}$ as leaves and the entire dataset X as root. This tree must then be 'sliced' at some appropriate height to yield the desired partition $\{\mathbf{x'_i}\}$. Finding a good height yielding a desired compromise between enough simplification (clustering) but sufficient presence of the details is not trivial.

Let us first consider an given level of partition of X into k clusters $\mathbf{x'_i}$, *i.e.*, $X = \bigcup_i \mathbf{x'_i}$. A key issue for both top-down and bottom-up clustering is how to define the similarity function δ' at the level of groups, based on a given similarity function δ defined on observations. For this, one typically chooses from three options, called single-linkage, complete (or full) linkage, and average linkage. For single linkage, $\delta'(\mathbf{x'_i}, \mathbf{x'_j})$ is defined as the minimum of $\delta(\mathbf{x_i}, \mathbf{x_j})$ over all elements $\mathbf{x_i} \in \mathbf{x'_i}$ and $\mathbf{x_j} \in \mathbf{x'_j}$, respectively. For full complete-linkage, $\delta'(\mathbf{x'_i}, \mathbf{x'_j})$ is defined as the maximum of $\delta(\mathbf{x_i}, \mathbf{x_j})$ over all elements $\mathbf{x_i} \in \mathbf{x'_i}$ and $\mathbf{x_j} \in \mathbf{x'_j}$, respectively. For average linkage, $\delta'(\mathbf{x'_i}, \mathbf{x'_j})$ is defined as the average of $\delta(\mathbf{x_i}, \mathbf{x_j})$ over all elements $\mathbf{x_i} \in \mathbf{x'_i}$ and $\mathbf{x_j} \in \mathbf{x'_j}$, respectively. Generally, average linkage is preferred, as it is less sensitive to data outliers or noise [40].

Fuzzy C-Means: In fuzzy clustering approaches, each data observation can belong to multiple clusters. To each membership association between an observation $\mathbf{x_i}$ to a cluster $\mathbf{x'_j}$, there is a weight $w_{ij} \in \mathbb{R}^+$ of this membership. This way, it is possible to observe how strongly an observation is associated to a given cluster, *i.e.*, transcend the limitations of 'hard' cluster computations such as given by K-Means, DBScan, or Hierarchical. One commonly used algorithm in this category is the *Fuzzy C-Means* technique [50]. Other techniques include algebraic multi-

grid [103, 72]. For the fuzzy C-means technique, the first step is to select the number of clusters k to partition the dataset into. Next, a matrix $U = (w_{ij})_{i,j}$ is initialized to hold the membership degree w_{ij} of each observation $\mathbf{x}_i$ to each cluster $\mathbf{x}'_j$. Then, an iterative procedure starts, computing the centroid for each cluster. Based on the newly computed centroids, the values in the matrix U are updated. This procedure finishes until convergence, *i.e.*, when changes of values in U become smaller than a predefined threshold. In mathematical terms, this procedure is equivalent to finding a basis $\mathbf{x}'_j$ that describes the observations $\mathbf{x}_i$ via the weights w_{ij}. Fuzzy clustering results can be visualized in a simplified way by depicting the clusters having maximal weights for each observation [72]. Visualizing the full set of fuzzy relationships is hard, since each of the N observations has k relations (weights) with all existing k clusters.

2.3 Multidimensional Data Visualization

Given our research question outlined in Chapter 1, we are interested to see how multidimensional datasets can be visually depicted, so that similarities between (groups of) observations become apparent. From this perspective, we can classify the existing multidimensional data visualization techniques into two groups:

- *observation-centric:* These techniques focus on depicting the observations explicitly. Various forms of relations between observations are captured in the visualization up to different degrees, but the focus is here on depicting the absolute observation values rather than their interrelationships. These techniques are briefly overviewed next in Sec. 2.3.1;

- *relation-centric:* In contrast to the above, these techniques focus (more) on the depiction of relations between observations. Absolute values of the observations may or may not be displayed along, but the focus is on the inter-relationships of observations rather than the observations themselves. These techniques, more central to our scope, are discussed in Secs. 2.3.2 and, further on, in Sec. 2.4.

2.3.1 Visualizing Multidimensional Values

Table Lens

A first, and arguably one of the simplest, techniques to visualize multidimensional observations follows from the table organization of multidimensional data outlined in Sec. 1.1. Here, given a dataset $X \subset \mathbb{D}^m$ of N observations, we can visualize it by drawing a table of N rows and m columns, where every row represents and observation and every column represents a dimension, respectively.

For relatively small datasets, in terms of N and m, individual attribute values x_i^j can be represented as text values in the table cells, such as in a standard Excel

sheet (Fig. 2.1a). However, when N and m surpass the values of what can be displayed in a readable form on s typical computer screen, other means are needed. One such mean is proposed by *table lenses* [150]: Here, each row is reduced to a single horizontal pixel line, formed by line segments which encode, by means of color and/or length, the cell values x_i^j (Fig. 2.1b). As such, columns are reduced to color-coded bars and/or to classical 1D bar charts. Subsequent sorting of rows on the value of a user-selected column (dimension) of interest can easily reveal patterns such as correlations, inverse correlations, or independence of the present dimensions. Further on, if the data values in $\mathbb{D}$ admit a simple aggregation, as is the case *e.g.* for continuous data values (Sec. 2.1.1), such a so-called *table lens* can be easily zoomed out to show an unbounded number of rows (observations). Additional cue such as shaded cushions can be used to *e.g.* delimit groups of similar-value observations in the sorted dimension [182].

a)   b)
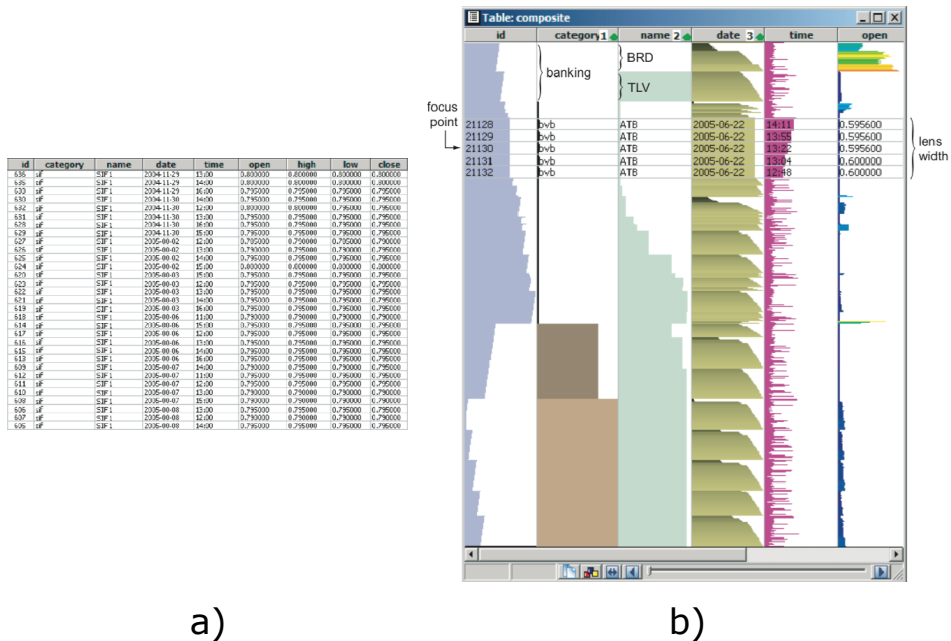
Figure 2.1: (a) Classical visualization of tabular data. (b) Table lens visualization. Images generated with TableVision tool [182].

A salient feature of table lenses is that, when combined with multiple dimension sorting and clustering of observations along the values of the sorted dimensions, they can easily create data hierarchies from unstructured data tables. In particular, multiple such hierarchies can be created from the same single dataset, depending on the order of sorting of data dimensions and the grouping criterion. This effectively yields a 'visual' way to apply operations equivalent to SQL database *GROUP BY* and *ORDER BY* statements. This allows the on-the-fly creation of a wide set

of hierarchies from the data, which can be next visualized using classical hierarchical attributed tree visualization methods, such as treemaps [198, 182]. Such mechanisms are quite powerful in creating multidimensional data views where observations are clustered in terms of subsets of attribute values. However, they also require a non-trivial amount of user interaction to construct these hierarchies from the data.

Scatterplot Matrices

A different approach of visualizing multidimensional data is to explicitly focus on showing the correlation of dimension pairs. Given, again, a dataset $X \subset \mathbb{D}^m$, the idea here is to show, for any pair of dimensions $1 \leqslant i \leqslant m, 1 \leqslant j \leqslant m, i \neq j$, the correlation of observations with respect to dimensions i and j. This yields a matrix of $m \times m$ cells, where each cell shows the correlation of x_k^i, x_k^j, for all observations j and the dimension pair (i, j). Scatterplot matrices (SPLOMs) are quite scalable in the number of observations, since each observation becomes a single point in a SPLOM cell. However, they are less scalable in the number of dimensions m, since a SPLOM grows quadratically with the number of dimensions m. More problematically, it is quite hard to reason about individual observations (and their similarities), since a single observation becomes in essence m^2 points in a SPLOM, one point per SPLOM cell.
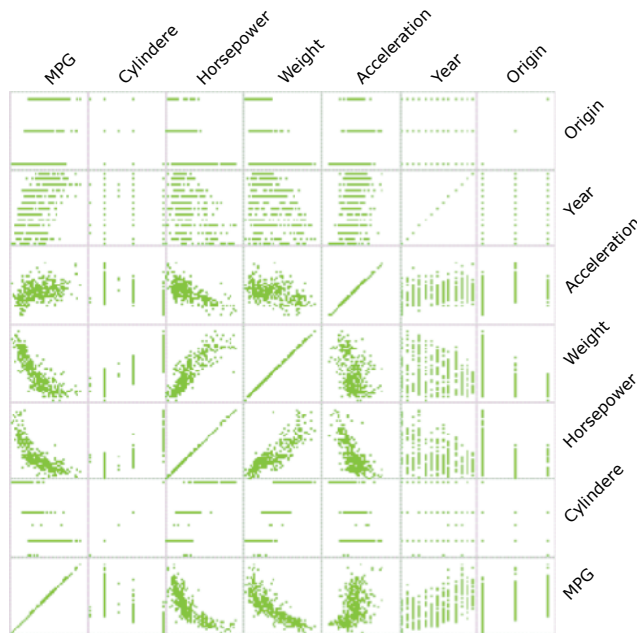

Figure 2.2: Scatterplot matrix of a 7-dimensional dataset.

Scaterplot matrices are effective, as stated, for a relatively small number of dimensions (roughly, $m < 10$). And, even for such relatively low dimensional datasets, they can be hard to interpret, since, as said a single observation becomes m^2 points in the SPLOM (or, more foemally speaking, $m^2/2$ points if we consider that the SPLOM is a square symmetric matrix).

Figure2.2 shows a SPLOM for a 7-dimensional dataset where each observation describes a car along the attributes miles-per-gallon (inverse of fuel consumption), number of cylinders, horsepower, weight, acceleration, year of manufacturing, and country of origin. The dataset is a standard benchmark for multidimensional information visualization techniques [183]. As visible, this is a hybrid dataset, including quantitative, integral, and categorical data. Several cells in the bottom-lower part show clear inverse correlation patters, such as between the miles-per-gallon (MPG) and horsepower.

Parallel Coordinates

Parallel coordinates is the third and last observation-centric visualization method for multidimensional data. Here, a (typically vertical) axis is drawn for each of the m dimensions. An observation $\mathbf{x}_i$ is next plotted as a fractured polyline of $m-1$ segments that connect the linear mapping of the values x_i^j along the m vertical axes. This way, similar observations become closely spaced polylines; and inversely correlated dimensions show up as characteristic x-like patterns of line segments linking dimensions which are mapped adjacently to each other in the plot. Observation values are easily visible by following where these fractured polylines intersect the m vertical axes. However, parallel coordinate plots suffer from several limitations: (a) They can generate a significant amount of line clutter for large datasets containing many uncorrelated observations; (b) they require a good, typically manual, arrangement of the dimension so that neighboring axes encode variables whose (lack of) correlation is interesting to see; and, last but not least, they are in general perceived as quite unintuitive, and require a non-negligible training of their users to become usable.

Figure 2.3 shows a parallel coordinate plot (PCP) for the same 7-dimensional car dataset as discussed in Sec. 2.3.1. Several enhancements atop of the basic PCP design are also illustrated here. Axis directions (top to bottom *vs* bottom to top) can be swapped to minimize the amount of line crossings. Ranges along the axes can be selected (see orange selection box on second axis from right in Fig. 2.3), to highlight the observations having the respective value range (red lines in the figure). This allows to better spot outlier observations, such as the red line diverging from the tight red-line bundle in Fig. 2.3.

As we see from the review of the standard multidimensional visualization methods, such methods can show patterns related to similarity in multidimensional datasets, such as groups of similar observations and outliers. However, finding such patterns can be hard for some of the visualizations, such as table lenses and
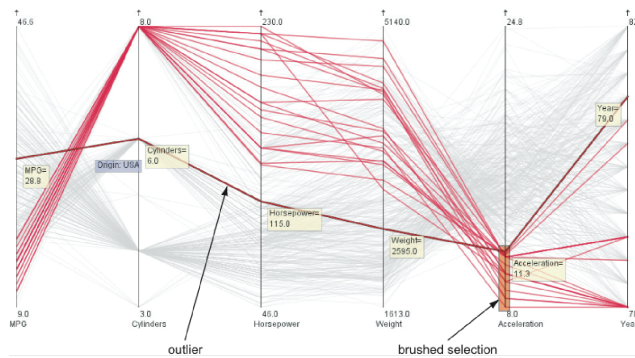
Figure 2.3: Parallel coordinate plot of the same 7-dimensional dataset as in Fig. 2.2. Image taken from [183].

SPLOMs, and arguably requires quite some training for PCPs. More importantly, none of the discussed visualizations scales well to over roughly ten dimensions.

2.3.2   Visualizing Multidimensional Relations

As explained in Chapter 1, our focus is on visualizing similarity in multidimensional datasets, and this task is intimately related to visualizing hierarchies and relationships in general. As such, we next provide a brief overview of visualization methods that focus on data that is both relational and multidimensional.

Diagrams

One frequently met form of multidimensional and relational data comes in the form of attributed graphs. In these, the graph part encodes the relational information in terms of nodes and edges; the multidimensional aspect is present in the form of several attributes added to nodes, edges, or both. An example of such datasets are multivariate networks extracted from software data, such as dependency graphs [44]. Here, nodes are software entities, such as functions or classes. Edges indicate interaction and collaboration between entities, such as calls or data flows. Attributes can be recorded both on nodes, *e.g.*, name, number of lines or code, and number of arguments of a function; and on edges, *e.g.*, duration of a function call.

   Probably the best known metaphor for the visualization is provided by *diagrams*. Here, the relational (graph) information is displayed using the classical node-link metaphor, and the multidimensional attributes are added atop of this skeleton by encoding it into glyphs, color, or texture. The positions of nodes in the visualization, or *embedding* of the graph, are computed using specialized graph layout algorithms [43]. Figure 2.4 shows several flavors of diagrams for the visualization of multivariate graphs describing the structure of software systems. As
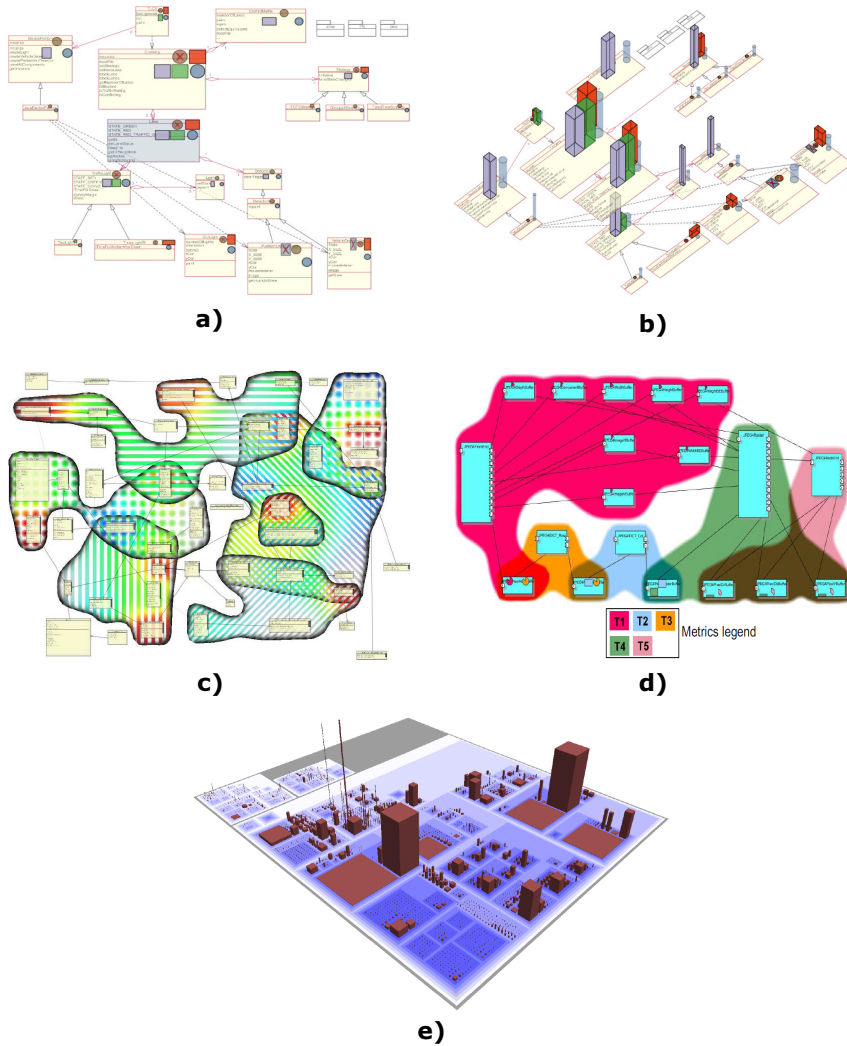
Figure 2.4: Visualization of multivariate attributed graphs using diagrams. (a,b) UML class diagram with software quality metrics ,[189]. (c,d) UML class diagram with software quality metrics and groups showing similar elements [22]. (e) Software system hierarchical structure with software quality metrics [205].

visible, graphs are drawn in two dimensions using a UML-like layout (Figs. 2.4a-d) or a treemap showing the containment of entities (Figs. 2.4e). Attributes are defined on entities (nodes) and are displayed using glyph shapes (Fig. 2.4a), glyph sizes (Figs. 2.4b,d,e), and glyph colors (Figs. 2.4a-d).

Diagrams are quite powerful tools for visualizing multivariate attributed graphs, mainly because they use a familiar visual metaphor. However, displaying multiple dimensions simultaneously is quite hard. These dimensions are shown either by encoding them into separate visual variables, such as size, texture, shape, and color; or by using a small multiple design, where each dimension is encoded into a different glyph. The first solution is limited to the small number of independent visual variables we can encode in the same image [12, 13]. The second solution allows up to about 20 variables to be displayed per entity (observation), as shown in the small table-lens-like displays drawn atop of the entities in the UML diagram in Fig. 2.4d. However, finding similar entities is very hard with this metaphor, as one needs to compare several tens of glyphs drawn atop of different entities in the diagram. To assist this task of finding similar elements,
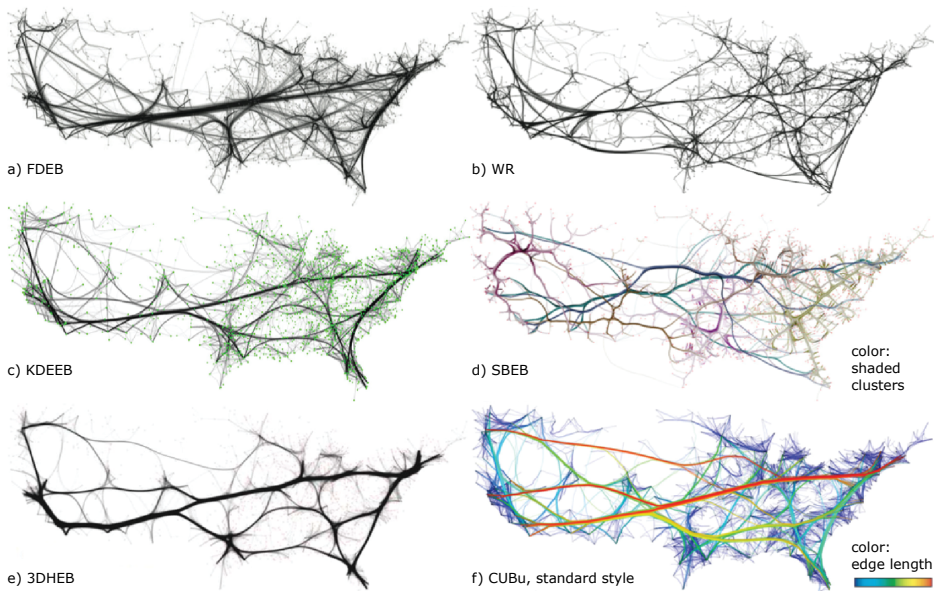
Figure 2.5: Graph bundling techniques: a) Force-directed edge bundling (FDEB, [84]). b) Winding roads (WR, [108]). c) Kernel density estimation edge bundling (KDEEB, [87]). d) Skeleton-based edge bundling (SBEB, [56]). e) 3D histogram edge bundling (3DHEB, [125]). f) CUDA universal bundling (CUBU, [194]).

Graph bundling

The node-link metaphor used for creating diagrams works reasonably well for graphs having hundreds up to roughly one thousand elements (nodes and/or

edges). Specifically, it is possible to reason about structural patterns by inspecting how nodes are positioned in the visual space and by examining groups of densely connected points. However, this metaphor can easily break down for larger graphs, due to the large amount of clutter created by overlapping nodes and/or edges intersecting at various angles. In such cases, seeing salient patterns in the graph structure can be very hard or even impossible, even when such patterns do exist.

This problem can be attacked by edge bundling techniques. They can simplify the visualization by aggregating and bending closely related edges, thus providing a coarser view of the structure of the same graph and drastically reducing the visual clutter. More specifically, edge bundling trades overdraw of edges (which is increased) for a diminished number of edge intersections happening at various angles. In more formal terms, if we consider the edge spatial density in the drawing ρ and the distribution of edge crossing angles δ, edge bundling 'sharpens' the signals ρ and δ [87]. This creates larger amounts of white space in the drawing, thus allowing one to better separate edge bundles from each other than original unbundled edges from each other, due to the sharpening of ρ; and makes edges in a bundle largely run parallel to each other, due to the sharpening of δ. Overall, bundling can be seen as a kind of coarsening operator, conceptually similar to the clustering methods outlined in Sec. 2.2, that operates on the drawing of a graph. Following this analogy, if an unbundled graph drawing is useful in assessing the node-to-node connections, a bundled drawing is useful in assessing the connections between groups of closely placed nodes [194].

One of the first, and arguably one of the best known, bundling techniques is Hierarchical Edge Bundling (HEB) [82]. This technique uses a hierarchy defined atop of the graph nodes to guide the bending and grouping of the edges. Edges are modeled as B-Spline curves, whose shapes are controlled by intermediate points of the given hierarchy. Bundling techniques have also been proposed for general graphs that do not avail of a node hierarchy [84, 35, 108]. Recent efforts in graph bundling have led to algorithms that scale well computationally to graphs of hundreds of thousands to millions of edges [64, 87, 194] and rendering techniques that emphasize the simplified structure of the bundled graph [185, 56]. A recent overview of graph bundling techniques is given in [213]. Additional related work on graph bundling is discussed in context in Chapter 4.

A relatively less well covered area of graph bundling is the treatment of multivariate graphs. In such graphs, nodes and/or edges have additional data attributes. This makes them effectively be multidimensional and relational datasets. For such graphs, one would naturally like to extend bundling to incorporate not only similarity of the graph *structure* (which is already captured by classical bundling techniques), but also of the graph *attributes*. This way, bundles would depict connections between groups of nodes which are similar both in terms of position in the layout and position in the attribute space. In this respect, directional bundling incorporates bundling of edges controlled by their direction, which supports the exploration of directed graphs [165]. Recently, this method has been extended

to also incorporate one quantitative attribute per edge when computing the edge similarities that drive the bundle formation [147]. Ersoy *et al.* mention that their method [56] can handle edge similarities driven by multiple edge attributes, but do not present examples hereof. Overall, the use of several attributes of both nodes and edges to create simplified views of multivariate relational datasets has not been explored further, to our knowledge.

Point-based tree drawing

As outlined earlier, the visualization of large datasets is typically approached by defining a multiscale representation, usually computed by means of clustering similar items. This applies also to multidimensional datasets. As such, hierarchies that capture the multiscale representation of the data can be used to increase visual scalability.

To allow the visualization of hierarchies from thousands of elements in a single single view, Schulz *et al.* [162] proposed an approach to visualize a tree using a modified point sampling approach. The approach uses a modified version of the $\sqrt{5}$-sampling algorithm [173], which was initially proposed to accelerate the rendering of polygons. To visualize trees, the approach in [162] uses a recursive strategy to position observations from the hierarchy. The first step is to define a grid in the visual space. Next, this grid is rotated in 27 degrees. The tree root is positioned in the center of the visual space, and its first four children are positioned along the grid diagonal (Fig. 2.6a). In the next iteration, the grid cell size is scaled down by a ratio $\sqrt{5}$ and the next four children of the root node are positioned along the new diagonals. In this same iteration, the four children of the previously positioned nodes are also placed (Fig. 2.6b), following the same pattern of the root node. This procedure is then repeated, by dividing the grid again, and positioning another hierarchy level (Fig. 2.6c), until all nodes are placed. In the final image, the depth of nodes in the tree can be color-coded so as to get a clearer impression of the tree structure (Fig. 2.6d).

Overall, the idea of this type of methods is to densely fill the available screen space by the tree structure, so that wasted white space is minimized, and also so that related data elements (children of the same subtree) are placed close to each other. Interaction techniques are added to aid the tree exploration. By *zooming* it is possible to focus only on a selected subtree, which is then scaled to occupy all the visual space and allow one to see more details. *Filtering* allow hiding tree regions that do not met user-defined search criteria, like visualizing only nodes from a defined level, or visualizing nodes that have a minimum number of children. *Rotation* allows that branches positioned in regions of less importance are "promoted" to more visible regions (Fig. 2.7).

Figure 2.8 shows a visualization from the hierarchy of DMOZ[1] websites. This hierarchy has $754,403$ elements and $576,818$ leaf nodes. Color maps the tree depth.

---

1 http://www.dmoz.org

(a) Placement of the root node and its four first children

(b) First division by $\sqrt{5}$, rotation, and children positioning

(c) Next divisions, grid rotations, and children positioning
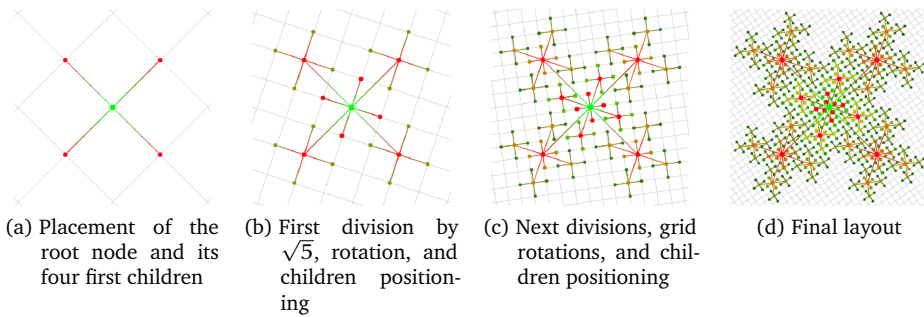
(d) Final layout

Figure 2.6: Overview of point-based tree visualization. The root node is positioned in the center of the visual space, and the remaining hierarchy is positioned in a rotated grid that covers the visual space.

(a) Original tree

(b) First rotation

(c) Second rotation
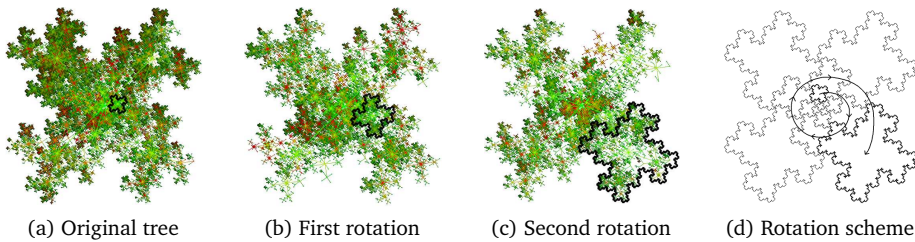
(d) Rotation scheme

Figure 2.7: Point-based tree rotation procedure. On every rotation, branches in regions of less importance are repositioned. Source: [162]

The authors have also assessed the severity of overplotting in their proposal by using a heat map (Fig. 2.8b), where blue indicates no overplotting and red indicates maximum overplotting, respectively. As visible, the technique achieves an overall quite low overplotting distribution.

The key advantages of this technique is its visual scalability that can easily show hundreds of thousands of nodes in a hierarchy simultaneously on the same screen. The technique also allows modifications in a hierarchy branch to remain local to the nodes positioned in that branch. However, the technique cannot be used to visualize node similarities encoded by continuous distance metrics – in other words, the visualization maps the position of nodes in the hierarchy, but cannot take into account edge weights. In contrast, classical graph layouts depicted with node-link metaphors can easily incorporate such constraints when placing nodes in the embedding space.

Multiscale Document Map

Another approach to visualize similarities between multidimensional observations was proposed by Nocaj and Brandes [131]. In their work, a multiscale visualiza-

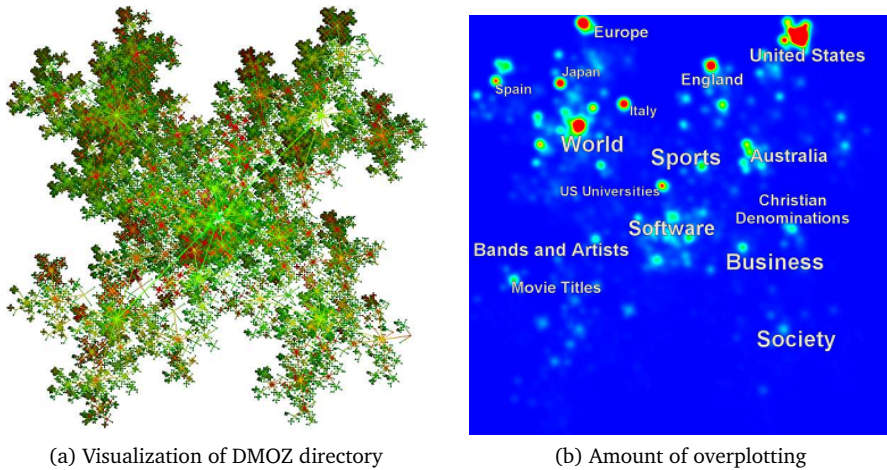(a) Visualization of DMOZ directory   (b) Amount of overplotting

Figure 2.8: Visualization of the DMOZ directory. The hierarchy contains 754.403 elements. The splatting view indicates regions with nodes overplotting. Source: [162]

tion showing similarities between documents is proposed to allow finding groups of highly related documents. To do this, they employ a multidimensional scaling (MDS) technique to project the documents to a 2D visual space, based on the preservation of similarities between features extracted from the documents. The MDS result is visualized using a scatterplot-like metaphor. To reduce visual clutter, they propose a multiscale approach that displays only a fraction of the total number of observations during the interactive exploration. To select which points are to be displayed, the document collection is clustered to create a hierarchy, and a single selected hierarchy level is displayed at a time, aiming to optimize the scatterplot density (number of displayed items taken from the current hierarchy level per visual-space area unit).

Furthermore, groups of similar observations, which are mapped by MDS to closely spaced points in the scatterplot, are emphasized by using Voronoi treemaps [132]. These also serve as visual cues that help maintaining the mental map when the user navigates between hierarchy levels.

The first step to build this map is to compute similarities between documents. For this, documents are converted to the Vector Space Model [159] where each document is represented as a vector of word (term) frequencies. Inter-document distances, computed using the cosine distance metric between vectors, are next stored in a distance matrix. Finally, this distance is used by MDS to place points representing documents in the visual space. More details over MDS are given next in Sec. 2.4.1. Separately, the document dataset is hierarchically clustered, based on the same distances, to yield a hierarchy organizing documents by similarity. The hierarchy is used both for selecting the desired level of detail (as outlined earlier), and also visualized using Voronoi treemaps.

(a) Reference map   (b) Map filtered by search   (c) Final layout according to the user search terms
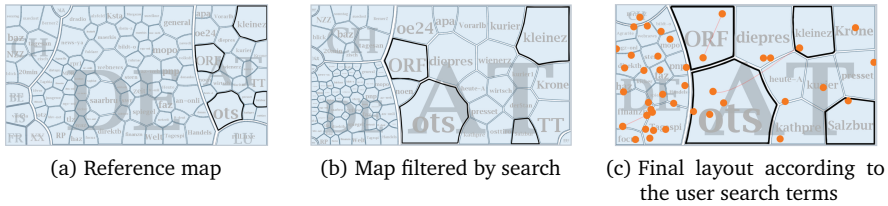
Figure 2.9: Multiscale document map. a) Reference map. b) As the user enters search terms, the map shows only documents that contain these terms. c) Adaptation of the map to emphasize document relevance with respect to the search. Images from [131]

The multiscale aspect of the visualization is also outlined by its interactive use. One starts with a so-called 'reference map' that shows a coarse level of the document hierarchy (Fig. 2.9a). As the user performs searches using keywords, the map is filtered to show only documents (and their surrounding cells) that match at least one of the entered terms (Fig. 2.9ab). To preserve the user's mentap map during such interaction, and diminish the abrupt changes appearing when switching between the display of different hierarchy levels, MDS is re-executed when the points to be displayed change, but the final layout of the displayed points is constrained to also follow their positions in the reference map. Documents that contain many hits of the search terms are emphasized by displaying them as larger points (Fig. 2.9ac). Finally, the similarity of the visualized points is emphasized by linking these by straight lines in the visualization – a technique which is reminiscent of the drawing of similarity trees [162] discussed earlier in this section.

GMap: Compact maps of point layouts

Given an embedding of a graph or multidimensional dataset in two dimensions, the GMap technique [63] proposes to visualize these, and highlight groups of similar items, by using a cartographic map metaphor. Here, groups of closely related points appear as countries; empty space separating such groups appears as rivers or seas; outlier points show up as islands. While the aims of this visualization are quite close to the ones discussed earlier – helping users to locate groups of similar observations – the proposed visual encoding makes the detection of groups easier, and also reduces visual clutter that appears in some other techniques, such as multidimensional projections or point-based tree layouts.

GMaps are constructed as follows. First, clustering is used to partition the input dataset into groups of similar points, based on any user-specified distance metric between observations. As clustering techniques, GMaps proposes to use K-Means or Fast Greedy [127], though other techniques can be used equally well. Next, a Voronoi diagram is constructed to create one cell per identified group (Fig. 2.10a). However, such cells can be arbitrary convex polygons, which do not necessarily

(a) Initial Voronoi diagram

(b) Voronoi cell smoothing
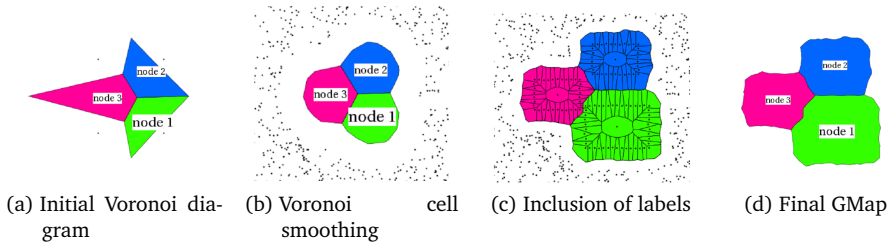
(c) Inclusion of labels

(d) Final GMap

Figure 2.10: GMap construction pipeline.

surround the points in an intuitive manner. This has been observed also by other related applications where Voronoi cells are used to visually group points in a multidimensional projection [18]. To alleviate such problems and make cells look more like regions on a cartographic map, Voronoi cells are smoothed by inserting a number of randomly placed artificial points around each data point and recomputing the diagram (Fig. 2.10b). Next, explanatory labels are defined for each cell, and placed in the cells themselves. To accommodate for this, the artificial points are arranged to follow the labels' hulls (Fig. 2.10c). Finally, cells corresponding to each initial group are merged to yield the final regions in the map (Fig. 2.10d).

(a) Scientific colaboration network

(b) Map defining main groups
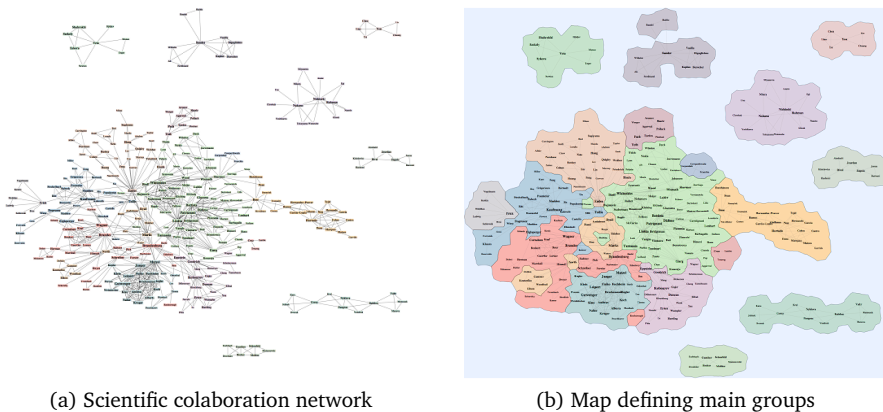
Figure 2.11: Graph and respective GMap of scientific colaboration.

Figure 2.11 shows an example of a GMap created to visualize a graph that encodes a scientific collaboration network whose nodes are 509 authors edges are 1,517 co-authorship relations respectively. Showing only the graph highlights a large central cluster of interconnected authors and a few outlier clusters (Fig. 2.11a). In contrast, the GMap splits the large cluster into smaller regions, and thus emphasizes collaboration patterns at a finer scale (Fig. 2.11b). In the same time, the outlier groups become easier to separate from the surrounding space. We shall

exploit a conceptually similar visual metaphor in our own work further described in Chapters 5 and 6.

2.4 Multidimensional Projections

Multidimensional Projection (MP) techniques are a central element of our research goals outlined in Sec. 1.3. Formally put, given a dataset X embedded in n-dimensional space, for instance in $\mathbb{R}^n$, MPs are are functions $P : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where $m < n$ and typically $m \ll n$. The result of a projection technique, *i.e.* the set $\{P(\mathbf{x})|\mathbf{x} \in X\} \subset \mathbb{R}^m$ is next denoted by X_P. In the literature, this set is sometimes also called a projection, which may create confusions between projection *techniques* (the function P) and the *result* of applying such a technique (the set X_P). Moreover, the value $P(\mathbf{x} \in X)$ is also called the projection of observation $\mathbf{x}$. In the following, we will next use the shorthand 'projection' to refer to all the above concents when the distinction is clear from the context, and otherwise refine the explanation as needed.

An additional property of MPs is that they aim to preserve the similarity structure of the data from X to X_P. While a formal and fully covering definition of 'similarity structure' is lacking, it is generally accepted that this takes two forms in practice. First, structure can be defined in terms of the pairwise distances $d^n : X \times X \rightarrow \mathbb{R}^n$ between observations in the input dataset and the pairwise distances $d^m : X_P \times X_P \rightarrow \mathbb{R}^m$ between the projections of the same observations. In this case, a projection technique is said to *preserve distances* if $d^n(\mathbf{x}_i, \mathbf{x}_j)$ and $d^m(\mathbf{x}_i, \mathbf{x}_j)$ are (nearly) identical up to a scaling factor for any point-pair $(\mathbf{x}_i, \mathbf{x}_j) \in X \times X$. Separately, structure can be defined in terms of the neighbors $v^n : X \rightarrow \mathcal{P}(X)$ and $v^m : X \rightarrow \mathcal{P}(X)$ of points in both the original dataset X and its projection X_P. Here, $\mathcal{P}$ denotes the power set; v^n is defined with respect to the distance d^n; and v^m is defined with respect to the distance $d^m(P(\cdot))$. A projection is said to *preserve neighborhoods* if $v^n(\mathbf{x}_i)$ and $v^m(\mathbf{x}_i)$ are (nearly) identical for any observation $\mathbf{x}_i \in X$. In the remainder of our work, we will consider projections from both types (distance preserving and neighborhood preserving). However, for the ease of exposition, we assume that the projections being discussed are distance preserving, unless otherwise specified.

Projections serve two main goals (see also Sec. 1.2). First, they can be used to simply decrease the dimensionality of a large dataset prior to its further processing, *e.g.*, by eliminating dimensions which are redundant, for instance due to a high correlation or low variance. Secondly, they can be used to get insights in the structure of a multidimensional dataset. Our research falls into the second context. For this, projections are typically used to embed high-dimensional data into 2D or 3D (*i.e.*, $m \in \{2, 3\}$), in which case the projected data X_P can be directly visualized by a scatterplot. If the projections preserves data structure, then one can use such a scatterplot to reason about the high-dimensional data patterns.

Projections can be classified as *global* or *local* techniques, based on which parts of the data structure they aim to preserve. Global techniques aim to preserve the structure in all spatial scales, *i.e.*, aim to preserve $d^n(\mathbf{x}_i, \mathbf{x}_j)$ for *all* point pairs $(\mathbf{x}_i, \mathbf{x}_j)$. Local techniques aim to preserve structure only on local (small) scales, *i.e.*, aim to preserve $d^n(\mathbf{x}_i, \mathbf{x}_j)$ only for low values of this distance. Overall, global techniques are technically simpler and arguably better when the task at hand involves comparing any point pairs. However, they generally succeed less in terms of highlighting patterns such as groups of strongly similar items. Local techniques are more complex, and also yield projections where far-away points may be incorrectly placed with respect to each other. However, they achieve far better results in terms of highlighting groups of similar items.

We next discuss several projection techniques in both the local and global classes.

2.4.1   Global Techniques

Arguably the best known (global) projection technique is *Principal Components Analysis* (PCA) [96]. The goal of this technique is to capture the principal directions in the $\mathbb{R}^n$ space that contribute most for data variance in X. In other words, the technique aims to find the m orthonormal directions that describe most of data variance. The solution for that is given by finding the m eigenvectors associated to the largest eigenvalues of the covariance matrix of X. PCA is simple to implement, fast to execute, and well known in the scientific and engineering literature [71, 70]. However, it has a major limitation: When the observations in X do not live on a hyperplane embedded in $\mathbb{R}^n$, the projection errors can be quite large. For instance, consider points uniformly distributed over the surface of the Earth. For this dataset, PCA will essentially project the points on a randomly oriented plane. As such, diametrically opposed points will falsely appear as being very similar.

Another global projection technique is *Self-Organizing Maps* (SOM) [102], which can be roughly seen as an extension of the K-Means clustering algorithm to project the high-dimensional data into a grid in the low-dimensional space, typically $\mathbb{R}^2$. This technique starts by firstly defining a number of labels (groups). Next, representative observations are randomly assigned to each cell of this grid, creating a Voronoi grid in the multidimensional space. The representatives of each cell are next updated using the average values of all observations that lie within a cell, thus defining a new arrangement for the membership of observations to cells. The process of updating representatives and assigning observations to cells continues until convergence. The final grid is then visualized in the target space, with observations placed in their assigned cells.

Another well known global technique is *Multidimensional Scaling* (MDS) [105]. It aims to minimize a so-called *stress* function, which computes how well the multidimensional reduction procedure preserved the distances from the high-dimensional space to the lower one. This function is essentially capturing the

ratio d^m/d^n discussed at the beginning of Sec. 2.4. Several approaches exist for this, many of them being based on minimization of the stress (also called error, or cost) function, *e.g.* by gradient descent [142]. To accelerate such optimizations, the LLE [155] and ISOMAP [188] methods use special numerical solvers for sparse eigenproblems. Another way for acceleration is to use a small set of so-called 'landmarks', which are observations in X that characterize well the overall structure of the dataset. Here, Landmarks MDS [168] and Pivot MDS [17] use classical MDS to place these landmarks in the target space and arrange the remaining points in X around the landmarks using simple interpolation schemes. Other techniques in the MDS class that achieve high computational performance are Fastmap [58] and MetricMap [204], and GLIMMER, which implements a multilevel MDS scheme on the GPU [89].

MDS projections are intimately related to computing graph layouts. Given the distance matrix $(d^n(\mathbf{x}_i, \mathbf{x}_j))_{i,j}$, one can construct a graph where the observations $\mathbf{x}_i$ are the nodes, and an edge $(\mathbf{x}_i, \mathbf{x}_j)$, with a weight $1/d^n(\mathbf{x}_i, \mathbf{x}_j)$, is present if $d^n(\mathbf{x}_i, \mathbf{x}_j)$ is smaller than some given threshold. Next, the graph can be laid out in $\mathbb{R}^m$ using a classical spring embedder technique [43]. This will place observations which are close in $\mathbb{R}^n$ close to each other in the embedding space. The positions of the graph nodes next give the desired projections $P(\mathbf{x}_i)$. The converse connection is also possible: Given a graph $G = (V, E)$ with nodes $V = \{\mathbf{x}_i\}$ and edges $E = (\mathbf{x}_i, \mathbf{x}_j)$, a distance matrix $(d^G(\mathbf{x}_i, \mathbf{x}_j))_{i,j}$ can be created, where $d^G(\mathbf{x}_i, \mathbf{x}_j)$ reflects the graph-theoretic distance between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Next, this matrix can be used to project the observations $\mathbf{x}_i$ to $\mathbb{R}^m$, by using any suitable projection technique. By drawing the edges in E long with the positions of the projected points $P(\mathbf{x}_i)$, we obtain a layout, or drawing, of the graph G [77, 119]. Overall, MDS methods are computationally scalable and easy to use. However, the faster such methods trade off precision (in data structure preservation) for speed, yielding results which may not be suitable for detailed visual analysis. Also, as the other considered global methods, they tend to have less power to separate local groups of similar observations when compared to the local techniques discussed next.

2.4.2   Local Techniques

As explained at the beginning of Sec. 2.4, local techniques attempt to preserve data structure only on small scales, *i.e.* for small distance ranges or neighborhoods defined around the high-dimensional observations $\mathbf{x}_i \in X$. This offers more flexibility in performing the embedding than global techniques. Indeed, depending on the characteristics of the data in a neighborhood, different parameterizations leading to an embedding, or even completely different embedding functions, can be used.

In the class of distance-preserving local techniques, most approaches use the earlier-mentioned idea of selecting a few landmarks for accurate positioning,

followed by fitting the remaining observations locally around their closest land-mark(s). For example, the *Least Square Projection* (LSP) [141] projects its land-marks by a force-based point placement procedure. The remaining observations are next projected using a Laplacian operator, which aims to position observations close to its representative neighborhoods and also achieve a smooth mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$. A faster local technique is the *Local Affine Multidimensional Projection* (LAMP)[95]. It first projects a small set of landmarks into the low-dimensional space and next interpolates the remaining instances using a family of affine map-pings. LAMP has the advantage of achieving high precision and performance, even using only a small number of samples. Another advantage of LAMP is that it al-lows direct control of the landmarks' positions in the projection, after which the placement of the remaining points is very fast. This allows one to interactively manipulate the projection, *e.g.* by moving the landmarks to better suit a desired visual organization or perceived inter-landmark distance. Several other local tech-niques exist [25, 146]. We use LAMP, a good quality, fast, robust, and easy-to-use projection in our work on attribute-based projection explanation in Chapter 5.

In the class of neighborhood-preserving local techniques, arguably the best-known one is the *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [193], which improves the earlier Stochastic Neighbor Embedding (SNE) technique [81]. These techniques first find a probability distribution that captures how likely is that an observation is the neighbor of another observation given a certain neighborhood size. A similar distribution is designed for the low-dimensional embedding space. Next, a cost function capturing the distance (difference) of the two distributions is minimized using gradient descent. This penalizes dissimilar observations in the high-dimensional space to be in close neighborhoods in the low-dimensional space – though, highly-similar observations in the high-dimensional space can become dissimilar in the low-dimensional space. The original SNE technique uses Gaus-sian distributions in both spaces. In contrast, t-SNE uses a heavy-tail Student t-distribution for the embedding space. Among other advantages, this makes t-SNE converge better and give better results in terms of neighborhood preservation. Recently, t-SNE was extended to also handle time-dependent datasets while guar-anteeing a good preservation of both spatial and temporal data patterns [151]. We will use dt-SNE in our work on visualizing multidimensional dynamic datasets in Chapter 6.

2.4.3   Assessing Projection Precision

From the previous discussions on projection techniques, it has become evident that their precision, defined in terms of their ability to preserve relevant data structures in the embedding space, is key to their usefulness for visualization pur-poses. Indeed, the 'inverse mapping' that connects a visual pattern discovered by the user in the visualization to a data pattern present, in our case, in the high-dimensional dataset X [187], can only take place correctly if the projection pre-

serves this type of pattern. Otherwise, two types of problems can appear. First, one may see patterns where actually none are in the data – a problem also called *false positives*. Conversely, one may never discover actual data patterns as these do not appear in the visualization – a problem also called *false negatives*.

Ideally, the precision of a projection should be measured as a function of the types of patterns it shows and their importance in a concrete task [164]. However, doing so it is quite hard, since such patterns can be very complex in general, and also it is hard to have ground truth for their evaluation, *i.e.*, high-dimensional data labeled to mark such patterns. As such, the precision of a projection is usually assessed by measuring how well it captures *similarity* relations, which are the basic building element for more complex patterns. Measuring the preservation of similarity can be done at different levels of detail. From coarse to fine levels, these are as follows.

Aggregated Normalized Stress

The stress function measures how well the projection function can preserve distances between all pairs of observations. Preservation is measured by a single aggregated value over all such pairs, which is also normalized for ease of interpretation across different projections. The aggregated normalized stress σ is defined as

$$\sigma = \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in X \times X} \frac{(d^n(\mathbf{x}_i, \mathbf{x}_j) - d^m(P(\mathbf{x}_i), P(\mathbf{x}_j)))^2}{(d^n(\mathbf{x}_i, \mathbf{x}_j))^2}. \tag{2.4}$$

Zero stress values give perfect preservation, while values larger than zero show increasingly poorer preservation. This formulation of aggregated normalized stress is the most used one in projection literature.

A slightly different formulation of the stress, called aggregated projection error, is proposed in [118] and is defined by

$$\epsilon = \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in X \times X} \left| \frac{d^n(\mathbf{x}_i, \mathbf{x}_j)}{\max_{i,j} d^n(\mathbf{x}_i, \mathbf{x}_j)} - \frac{d^m(P(\mathbf{x}_i), P(\mathbf{x}_j))}{\max_{i,j} d^m(P(\mathbf{x}_i), P(\mathbf{x}_j))} \right|. \tag{2.5}$$

This formulation is largely equivalent to the stress defined in Eqn 2.4, being however slightly more sensitive to outlier observations.

Aggregate measures are simple to compute and compact. However, as any aggregate metric, they only give a global indication of the overall quality of a projection. This is sufficient when statistically comparing several projection methods across a large collections of dataset to determine which is better. However, for a specific dataset, one typically wants to know (a) *where* in the projection (over which observations) do large errors appear; and (b) what *kind* of errors these are, *e.g.*, false positives or false negatives.

Neighborhood Preservation

The average neighborhood preservation (NP) metric aims to measure how much a projected layout preserves the neighborhoods of original high-dimensional observations. It is analogous to the average normalized stress for the situations where we are interested in reasoning about neighborhoods rather than absolute distances. The NP metric is computed as follows. Let $v^n : X \to \mathcal{P}(X)$ and $v^m : X \to \mathcal{P}(X)$ be functions that compute the neighborhoods of points in the original high-dimensional space, respectively the projection, as defined at the beginning of Sec. 2.4.

$$NP = \frac{1}{|X|} \sum_{x_i \in X} \frac{|v^m(x_i) \cap v^n(x_i)|}{|v^m(x_i)|} \tag{2.6}$$

A value of $NP = 1$ indicates perfect neighborhood preservation, while values $NP < 1$ indicate problems such as false positives and false negatives.

Obviously, the definition of NP is a function of the size of the neighborhoods being used. Typically, these are constructed using k nearest neighbors, so v^n, v^m, and NP are actually functions of k. As such, NP is usually visualized by plotting its values for all $k \in \{1, \ldots, |X|\}$. Here, the value k acts much like a scale factor on which one assesses the NP value. Interestingly, the plots $NP(k)$ are not monotonic in k. Indeed, for $k = 0$, $NP = 1$, since any point is its own neighbor; and for $k = |X|$, $NP = 1$, since a point will naturally contain all other points in X when the neighborhood size is the entire dataset. As such, assessing which ranges, or values of k, of the plot are the most relevant to study to capture the quality of a projection in terms of neighborhood preservation, is a nontrivial question. Neighborhood preservation metrics are used in Sec. 3.2.1 to compare various algorithms for constructing similarity trees.

Neighborhood Hit

The neighborhood hit (NH) metric is a variation of the NP metric used to determine the quality of group preservation for labeled data. To compute it, one proceeds by defining the same neighborhoods v^m and v^n as for the NP metric. Next, for each point $x_i \in X$, one computes the ratio of how many neighbors in v^m have the same class as x_i to how many neighbors of $P(x_i)$ in v^m have the same class as x_i. Again, values are averaged over classes, points, and neighborhood sizes, and the results are presented as a function $NH(k)$ of the neighborhood size k.

Local Measures

At the finest level of detail one can inquire about projection errors for specific subsets of points in a projection. For instance, one can measure the stress σ, aggregated error ϵ, neighborhood preservation NP, or neighborhood hit NH as func-

tions of observation $\mathbf{x}_i \in X$, by suitably adapting their aggregated formulations. This allows plotting such errors over the projection, and thereby finding specific zones where they appear. In the same way, measures can be designed not only to show the amount of errors, but which points cause these errors. For example, one can show which of the neighbors of a given $\mathbf{x}_i$ have been placed too close to $P(\mathbf{x}_i)$ in the projection – that is, show so-called *false neighbors*. Similarly, one can show which of the neighbors of a given $\mathbf{x}_i$, in the high-dimensional space, have been placed too far away from $P(\mathbf{x}_i)$ in the projection – that is, show so-called *missing neighbors*. The same notions can be extended to members of a group of close points, yielding notions of missing group members and false group members, respectively. All these local measures can be visualized by combinations of heat maps, level sets, projection triangulations, and bundled edges, leading to detail-rich and insightful descriptions of local errors [118, 120].

Compared to the aggregated metrics presented earlier, such local measures are less useful for deciding which of two projection techniques is in general better. However, they are useful for telling users where, in a projection, can one trust the visible patterns, and where not – thus, they serve as aids that guide the interpretation of a given projection for a concrete dataset [160, 118]. An adapted version of such local error measures has been proposed by Pagliosa *et al.* to color-code the value of quality measurements on a family of projections and their interpolation, to show differences between projections *vs* a particular error-distribution [135].

2.4.4 Explaining Projections

As we have seen so far, multidimensional projections are efficient and effective tools for mapping multidimensional datasets to 2D or 3D scatterplot-like views. A key ability they have, and which is essential given our research interest for exploring similarities of such data, is that they are geared precisely to show how similar an observation is to all other observations. However, the above ability is by itself not sufficient for making projections usable to explore data similarities. Two problems exist in this respect, both caused by the fact that a classical scatterplot view, used for a projection, shows just a point cloud, each point being an observation:

- *interpreting observations:* While each point $P(\mathbf{x}_i)$ in a projection represents precisely one observation $\mathbf{x}_i$ from the input high-dimensional dataset, the inverse mapping $P(\mathbf{x}_i) \rightarrow \mathbf{x}_i$ is not explicit. In other words, we do not know which observation $\mathbf{x}_i$ corresponds to a given point $P(\mathbf{x}_i)$ in a projection;

- *interpreting dimensions:* Compared to all other multidimensional visualization techniques reviewed in Secs. 2.3.1, projections are the *only* technique that does not explicitly encode dimensions. As such, projections may show that certain observations are similar, but do not tell why this is so.

To address both above issues, several so-called *explanatory* mechanisms have been proposed. Globally put, these enrich a raw projection scatterplot with ad-

ditional information that helps interpreting observations and/or interpreting dimensions. The most prominent such mechanisms are outlined next.

Interaction

Interactive techniques explain projections by showing on-demand information to help making sense of the group and outlier structures visible in the projection. Basic techniques include brushing close points in the projection to see which dimensions make them similar (this requires significant effort and memorization); and scagnostics methods which pre-analyze all scatterplots in a scatterplot-matrix (SPLOM) [10] to detect which ones best capture interesting patterns in D^n [192] (this still requires interaction and manual linked-view correlation). Other methods include ForceSPICE which uses a force-directed spring model to lay out a scatterplot of textual elements, on which the user can incrementally add annotations to highlight specific items [54]; and using phylogenetic trees to project documents by placing similar ones in close tree nodes [34]. Next, users can execute a topic extraction algorithm to automatically label selected tree branches to guide exploration. A technique suitably called 'explainers' create custom projection functions that align with user-specified annotations, by using machine learning optimizers [66]. Overall, all such interactive methods can explain regions in a projection D^m, but require user interaction effort to specify *where* to explain the projection.

Color Coding

Color coding is arguably the best known and simplest explanation for interpreting dimensions. Given one dimension x^j of the n dimensions of a dataset, points $P(x_i)$ in the projection are colored to indicate the values x_i^j. If dimension j is quantitative, a continuous colormap is typically used. If dimension j is categorical, a categorical colormap is used. This allows, at the lowest level, seeing how dimension j varies over the projected points. Furthermore, it potentially allows explaining the reason for appearance of patterns such as groups or outliers in terms of different value ranges of dimension j over such structures.

Figure 2.12 exemplifies this. Here, a projection of $1,000$ observations and 16 dimensions is color mapped to values of a selected dimension. A color legend on the top-left i shows that dark brown maps low values and bright yellow maps high values of the selected dimension, and that this dimension is quantitative. in the projection, we see a relatively well separated group of points to the left (surrounded by a dotted outline). We also see that all points in this group are dark, thus have low values of the selected dimension. Moreover, all points outside this group have higher values of the same dimension. Hence, we can say that the group can be explained as 'all observations that take low values in the selected dimension'.

While simple to understand, color coding has some strong limitations. In most real-world cases, groups and outliers in a projection cannot be explained by the
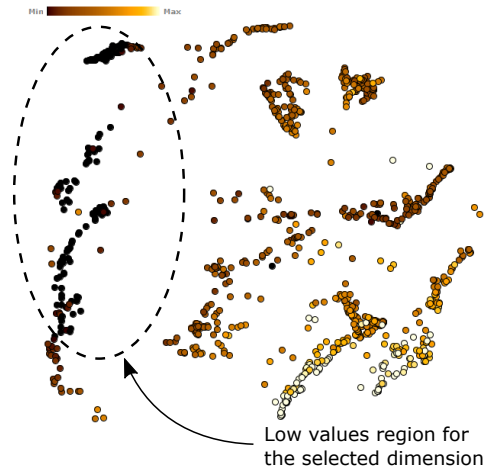
Figure 2.12: Color coding explanation of a multidimensional projection.

values of a *single* dimension. More formally, this can only be done when the same groups and outliers are also visible in the projection of the high-dimensional dataset on one of the planes defined by two axes of the $\mathbf{R}^n$ space the data resides in. Often, however, such groups and outliers can be explained by *sets* of dimensions, *i.e.*, combinations of different dimension values. For example, considering a projection where three groups are visible, it can be possible that one group is explained by certain values of dimension 1, the second group by certain values of dimension 3, and the third group by the remaining values of dimensions 1 and 3. Such insights can be obtained by successively color coding a projection by the values of all n dimensions, and mentally combining the impressions obtained from each such view. It is clear that this approach does not scale to more than a few dimensions, as it heavily relies on human memory.

Biplots and Axis Legends

A second class of explanatory mechanisms for interpreting dimensions is formed by biplots and axis legends.

Biplots are essentially a generalization of classical Cartesian coordinate axis shown in function graphs, for the context of projections. They work as follows. For an n-dimensional dataset, we have n orthogonal dimensions, or axes $\mathbf{a}^i$, $1 \leqslant i \leqslant n$. Data values range along each axis within an interval $[\mathbf{a}^i_{min}, \mathbf{a}^i_{max}]$. Biplot axes are essentially the mapping of the line segments determined by $[\mathbf{a}^i_{min}, \mathbf{a}^i_{max}]$ along all axes $\mathbf{a}^i$ to the low-dimensional embedding space via the projection function P. As such, they show, in the visual space, the directions of maximal variations of all n original variables, as well as the ranges of these variables. They can be used to judge the values of specific projected points along any dimension, much as it is done when viewing a Cartesian plot, except that they are usually not or-

thogonal to each other. Secondly, they can be used to judge the correlation of dimensions: If two biplot axes are nearly parallel, then the respective dimensions are strongly directly or inversely correlated, the correlation sign depending on the relative orientation of the biplot axes. If two biplot axes are nearly orthogonal, then the respective dimensions are uncorrelated. As such, a projection with biplot axes can be seen as a generalization of a scatterplot to a multidimensional dataset.

Biplots can be constructed by using SVD decomposition on the high dimensional dataset [197, 71, 70]. However, this works well only for linear projection techniques such as PCA. For non-linear projections, Coimbra *et al.* [28] generalized biplot axes by densely sampling the lines $\mathbf{a}^i$ in high-dimensional space, projecting the samples via the projection function P, and connecting these to yield the biplot axes in the low-dimensional space. Apart from the above-mentioned properties of biplot axes, this technique allows assessing the (local) non-linearity of the projection by looking at the curvature of the biplot axes. The example in Figure 2.13a illustrates these generalized biplot axes for a dataset containing 2814 elements from a dataset of 9 dimensions projected via LAMP [95]. The biplot axes intersect at the projection centroid, and labels are positioned next to each axis to indicate the mapped dimension by that axis.

(a) Biplots axis explanation. Image from [28].   (b) Axis legends explanation
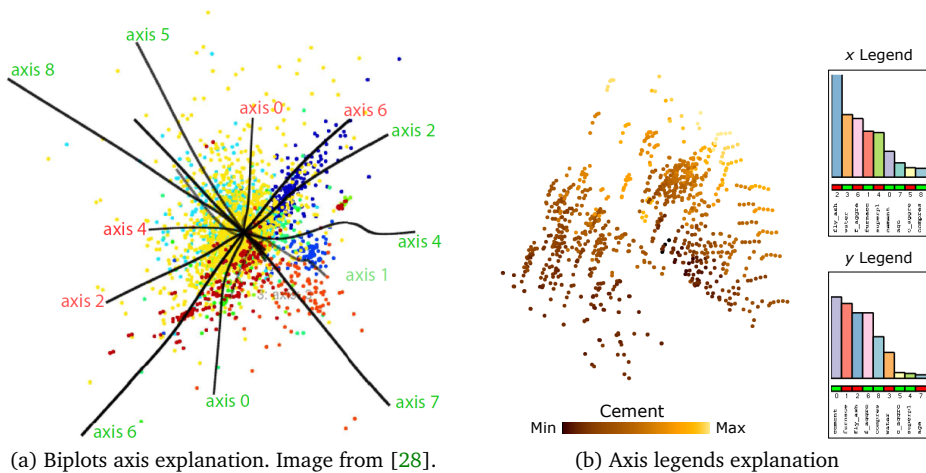
Figure 2.13: Biplots and axis legends explanation of multidimensional projections.

A second way to explain dimensions is to visualize how each of the n dimensions of the original dataset influences to the positioning of projected points along the screen axes x and y. In other words, this tries to explain what the x and y screen axes mean, in terms of the original n dimensions, rather than biplot axes which explain what the original n dimensions mean in terms of the screen x and y axes. To do this, one first observes that, for all but trivial cases, the x and y positions are explained by a mix of all n dimensions, each contributing up to different amounts. These amounts are called loadings for linear projections such

as PCA [71], and can be visualized by drawing a n-element bar chart for the x and y axes, where a bar length indicates the contribution of one of the original dimensions to the respective screen axes [18]. For other projection techniques, quantities similar in spirit to loadings can be computed by projecting the generalized biplot axes along the screen axes [28]. Further color coding and sorting of the bars allows users to tell which are the most important dimensions that determine the spread of points along the screen axes or, in other words, allows interpreting the projection much like a Cartesian scatterplot. Figure 2.13b illustrates axis legends for a projection of 1030 observations. This dataset epresents mixtures of ingredients required to make construction concrete and has 8 dimensions (ingredients) [210]. According to the axis legends, dimension *cement* and *furnace* are the ones with highest influence over the point mapping to axis y, whereas dimension *fly ash*. To verify this, one can color code the projected points by one of these dimensions. In our example, we do this on the values of the *cement* dimension. The observed dark-to-bright color gradient matches the vertical direction (y screen axis), which matches the fact that *cement* is the largest bar in the y axis legend.

Clustering and Labeling

All explanatory mechanisms presented so far share two characteristics: They are *global*, in the sense that they do not explain certain projection patterns by different variables than other patterns; and they are *dimension centric*, in the sense that they focus on showing dimensions and linking these with observations, rather than showing observations and linking these to dimensions.

An alternative way to explain a projection is to do this locally, focusing on explaining specific groups of points. These groups might be obtained both by manual selection in the visual space, or by an automatic clustering technique taking as input either the projected space or the original space. Paulovich *et al.* [143] propose such an approach to automatically explain projected groups from textual data (Fig. 2.14). For this, they segment visual groups in the projection space, and next compute the convex hull of each set of segmented points, or cluster. Next, documents in each cluster are analyzed to find relevant (high frequency) terms, which are then used as explanatory keywords for that cluster. The few most-relevant such terms are then drawn atop of the cluster using a tag cloud label layout, with labels scaled by the relevance of the respective terms. This way, clusters are explained by the most frequent data values that their observations take.

Other cluster-based explanation of projections exist. ImageHIVE extracts representative points from clusters of points representing images, and show these using graph drawing techniques [180]. Similarly, Nocaj *et al.* visualize documents by hierarchical clustering of the projected points and drawing cluster representatives [131]. Kandogan [99] visually annotates clusters in the embedding space based on the attribute trends detected in them. Clusters are computed by an

a)   b)

Figure 2.14: Local explanation of a projection layout. Based on a user-defined selection (a), regions of the projection are delimited. Next, each region is explained by the most frequent values its dimensions meet over all contained observations (b).

image-based scatterplot density estimation. Important attributes are found based on their statistical relevance.

Compared to earlier global explanations, local explanation has the strong added value of being able to adapt the explanation to the actual structure of the data. In other words, different groups of observations can be explained by different dimensions and/or dimension values, as needed. However, such explanations are, to our knowledge, mainly limited to text documents represented by a Vector Space Model, and do not generalize to other datasets such as quantitative ones. Moreover, the explanation relies on the precise delineation of meaningful clusters. We shall see in Chapters 5 and 6 how local explanations can be extended to overcome such limitations.

3

Multiscale Exploration of Similarity Trees

Similarity-based visualization techniques map observations onto a visual space such that similar ones can be recognized as such by the user. As explained in Chapter 2, similarity over a set of multidimensional observations can be visually encoded using either projection techniques [95, 193] or similarity-based layouts such as similarity trees [34, 141, 89, 136], classical treemaps [11], and Voronoi treemaps [132]. However, when collections involving hundreds of thousands or even millions of observations are given, few if any of the above methods are scalable both visually and computationally. In other words, for large datasets, such methods are either slow, or they generate clutter and mixture of potentially distinct data points in the final image.

To find potential solutions to the problem of visualizing similarity in large multidimensional datasets, let us further examine the advantages and limitations of our two classes of techniques of interest – multidimensional projections and similarity trees.

Multidimensional projections: These techniques are fast enough to allow the construction of similarity-based scatterplot-like maps for millions of observations. Moreover, recent techniques offer a good control of the preservation of similarities (distances) from the original high-dimensional space to the projection space, as explained in Chapter 2. However, there is little help in exploring the resulting maps at different *scales* or levels of detail: At coarser scales, points tend to group together. This impairs their visual separation and, in the limit, when one visualizes a large dataset with a limited projection (screen) space, one may even get a significant amount of clutter and overdraw. At finer scales, there is significantly less consistency among neighbors, since projection methods cannot fully preserve distances from any high-dimensional dataset to the 2D projection space [139]. In other words, when exploring a small region in the projection space, projection errors, present in the form of missing neighbors and false neighbors, become more apparent and may adversely influence the interpretation of the data [118]. These problems of projections have been recognized, and multiscale approaches for computing and exploring multidimensional projections have been proposed [140]. However, to our knowledge, such approaches do not scale computationally to handle large datasets, and also are geared to the visualization of textual data.

Similarity trees: Trees as encoders of similarity relationships [34, 136] allow the exploration of a visual map both globally and locally employing the same visual metaphor, as the tree's branch structure organizes similarity across the levels of a

tree. In other words, levels in a tree correspond to different degrees of similarity between nodes. This allows to directly find groups of similar items, which will be located on the same or neighbor levels in the tree. Besides this partitioning of the 'similarity space', levels also provide a multiscale description of the observations, with layers close to the tree root having fewer nodes that give a coarse representation of the dataset, and layers close to the tree leafs giving a finer-grained description. However, a limiting factor for similarity trees is that, to be constructed with good precision in terms of similarity-based grouping of nodes, they require $O(n^3)$ time and $O(n^2)$ memory for a dataset of n observations [169, 206]. This is prohibitively slow for large data collections.

In this chapter, we aim to create a visualization algorithm for similarity of multidimensional datasets that combines the advantages of multidimensional projections and similarity trees and reduces their disadvantages. For this, we propose a novel visualization approach to construct and explore similarity trees, based on a partitioning strategy which allows the exploration of large collections efficiently and with good precision concerning the assessment of similarity. Visual scalability is handled by making the visualization to reflect the scales of a recursive partitioning algorithm. The exploration strategies implemented with this scheme allow the exploration of large trees in varying levels of detail while still keeping a global perspective of the collection. We call this combination of techniques for large scale similarity visualization the *Visual SuperTree* (VST). We evaluate our approach on large collections of images, text and other data types. The VST enables exploration of large datasets at interactive rates and the multiscale approach for navigation allows fast drilling down to interesting data areas. In other words, VSTs provide a scalable approach to visualizing large multidimensional data both in computational and visual terms.

The remaining of this chapter is organized as follows. Section 3.1 presents related work on similarity-based visual mapping strategies. Section 3.2 describes the construction and exploration strategies of Visual SuperTrees. In section 3.3.1 we assess the precision of our VSTs on several datasets. Section 3.4 present use cases of VSTs for the exploration of large datasets. Section 3.5 discusses our technique. Section 3.6 concludes this chapter.

3.1   Related Work

As explained in Sec. 2.3, an important goal of multidimensional visualization methods is to reflect the similarities of the visualized items, defined in terms of either their data attributes or their relationships. The construction of visual maps based on similarity among observations usually relies on point placement strategies, also called embedding strategies, that encode similarity in different ways. We refine this topic below so as to better understand the existing techniques that address this goal.

Classical methods: In general, any (multidimensional) visualization method reflects the similarity of data items, up to certain extents, either implicitly or explicitly. Indeed, if the data-to-visual-variables mapping proposed by the method is consistent, then it should map identical data items to visually identical visual items; and should map highly similar data items to highly similar visual items [183]. The difference between methods appears in the extent up to which they satisfy this property, *i.e.*, how easy is to recognize data similarity by looking at the similarity of the visual items that map the data.

Classical multidimensional visualization methods achieve the above goal up to various extents. For instance, table lenses map similar observations (rows) to sets of horizontal bars having similar lengths and colors. However, due to the layout of the visualization, it is hard to find such rows, especially if they are not close to each other in the table [150]. Scatterplot matrices (Sec. 2.3.1) are, by excellence, a dimension-centric visualization, so they cannot easily show the similarity of observations. Detecting pairs of dimensions that have similar correlations is possible, but not entirely easy, as it involves finding cells in the matrix that show similar two-dimensional point clouds. Parallel coordinates map similar observations to sets of polylines that closely follow the same visual path (Sec. 2.3.1). While it is possible to visually find such sets, the task can be seriously impaired by clutter and crossings in case of larger datasets.

Multidimensional embeddings: These techniques are, arguably, the most used approach when one focuses on mapping observation similarity (Sec. 2.4). Recently, large progress has been made in building alternative projections of multidimensional data, searching to improve optimization criteria, computational complexity and precision, as well as local control over the results [141, 89, 95, 193]. Most such approaches are computationally fast and, subject to proper parameter choices, effective regarding the target goal of preserving distances in the original high-dimensional data space.

At the core of the interpretation of such point placements is the user's ability to recognize proximity in visual space, and to use this as an indication, or 'proxy', of detecting similarity or correlation of the original high-dimensional observations. This works arguably better than for other multidimensional visualizations – that is, it is easier to locate and separate compact groups of points in a 2D point cloud than, for instance, locate and separate bunches of closely-drawn polylines in a parallel coordinate plot. While we are not aware of formal analyses of the reasons behind this, it seems to use that this is due to the much stronger visual separation (of similar items from other items) that 2D point clouds offer in contrast to other mappings. Moreover, clutter, if present, does not impair separation in a 2D point cloud; one can even say that having more observations only increases the ease of separating groups of compact points (if these exist in the visualization, of course). In contrast, clutter in parallel coordinate plots, coming in the form of intersections between polylines, decreases the ease of visual separation.

In terms of input data, two alternatives are known for projections. First, embedding can directly use similarity (or distance) relationships between observations. A second alternative is to use the actual dimensions of the observations. Methods in the first class are by construction more versatile than methods in the second class, as they do not need actual dimensions but only distances between observations. However, such methods need to store a (typically large) distance matrix, and are also slower than the fastest embedding algorithms known, which directly use dimensions. Methods in the second class can be reduced to those in the first class, given that a distance metric can be computed from the dimensions. However, in some applications, one does not avail of explicit dimensions for observations, but only of distances between them.

Similarity trees: An alternative to classical embedding, similarity trees [34, 136] have been proposed to reflect similarity relationships. Such approaches organize the similarity relationship in levels. This allows the recognition of similarity patterns beyond those captured by 'flat' multidimensional projections. Another advantage of similarity trees is to capture, within a particular branch, observations sharing similar properties, due to the use of the precise Neighbor-Joining (NJ) phylogeny reconstruction algorithm, which is widely used in biology [65]. Additionally, compared to other point embedding strategies, visual clutter is reduced with similarity trees by the constraints of the underlying tree layout algorithm. Finally, the tree structure, visible in the form of branches, provides cues or 'reading paths' for the multilevel exploration of the embedding.

The precision of similarity trees in terms of capturing similarities was originally demonstrated for applications in visualization of collections of textual documents as well as for image collections by Eler *et al.* [52]. Similarity trees were also employed as a supportive tool for supervised dimension reduction processes [136]. A major disadvantage of a precise similarity tree is its construction time, which impairs computational scalability to large data sets. Another issue of similarity trees regards visual scalability: For relatively small trees, the underlying tree layout algorithm separates branches quite well from each other, leading to an uncluttered display. However, for datasets larger than a few thousand elements, such tree layout algorithms cannot prevent clutter and overlaps to form, thereby limiting visual scalability. Besides node-link layouts, similarity trees can be also visualized using space partitioning schemes such as treemaps [11, 132]. However, several such methods also have computational and/or visual scalability issues.

Our proposal: In this chapter we propose a new type of similarity tree, built to explore large multidimensional datasets. Our algorithm handles computational scalability problems by partitioning the data space and by creating a visual representation of the individual partitions that is based on the Neighbor-Joining (NJ) principles, described in detail next in Sec. 3.2.1. To achieve this, we pre-cluster the dataset in multiple levels, creating a set of recurrent NJ trees of decreasing sizes. Each such tree represents a level of the dataset's partition. These trees are

next joined to create a global, dataset-wide, similarity 'supertree'. This final tree is visualized via an exploratory interface that presents the data in a multiscale fashion, using an overview plus detail metaphor [27], by allowing the user to move up and down the levels of the tree to examine the data. This addresses the visual scalability issues mentioned earlier.

Compared to hierarchical clustering, both agglomerative (bottom up) and divisive (top down) [93], our approach also produces a multilevel solution that leads to a multiscale exploration. Specifically, by forming the final similarity tree by joining data clusters, the similarity relationships among clusters are captured in the tree, which next helps analyzing similarity at various levels of detail. In contrast to clustering, our approach does not has as end goal the creation of a tree that encodes similarity in a multiscale fashion, but the creation of a layout that preserves similarity relations in the final visualization.

The term supertree has also been used before in a different context within computational biology. In that context, it refers to the trees constructed by the combination of different phylogenetic trees for overlapping sets of taxonomic units [69]. The aim there is to combine trees built by different specialists and from different data, obtaining a single large consensual tree. Building biological supertrees is computationally hard when different input trees contradict each other with respect to specialization. Minimizing the disagreement among trees is also difficult [48]. However, such problems are not related to our problem here of organizing and displaying large quantities of observations by similarity.

Trees in visualization have been frequently used as a means to express multi-dimensional hierarchical data in various ways [161]. Many visualization systems allow effective ways of exploring trees via various versions of link-node representations [149]. Widely used techniques such as treemaps [11] improve space usage against conventional link-node representation and optimize the number of observations that can be presented in a rectangular space without occlusion. Treemaps have also been adapted to handle non-rectangular spaces [8], and also to improve the harmony and effectiveness of display [132]. Another example of this class of techniques is presented by Schulz *et al.* [162], who build a tree through rotation and grid-fitting procedures to fully occupy the visual plane. Although space-filling tree-display techniques have been designed to show data that are hierarchical in origin, they can be directly used on data which can be transformed into hierarchies, such as similarities.

While space-filling methods are very effective in terms of the information density they can realize for a given screen surface, they have the disadvantage – when compared to node-link tree visualization methods – of not being able to easily show the neighboring relations between tree branches. We believe that showing such relations is essential for interpreting similarity on a multiscale, and thus choose for a node-link visualization metaphor. This metaphor also reflects well the binary structure of the underlying NJ trees. Separately, our method is adaptable to uneven group distributions and also lends itself to the creation of multiscale mosaic views. Such views are very helpful when data can be meaning-
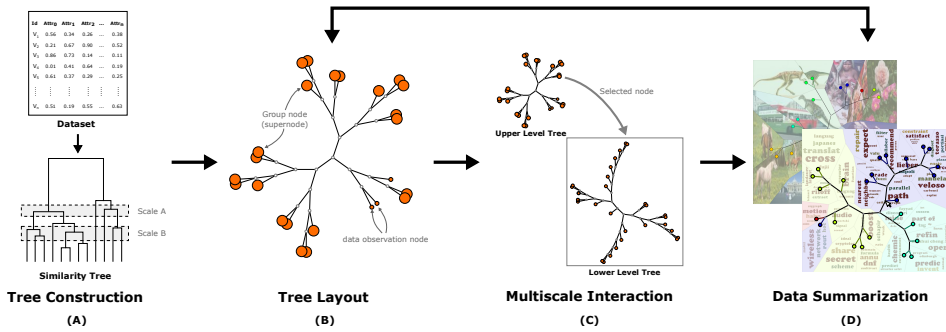
Figure 3.1: Overview of the Visual SuperTree construction pipeline.

fully summarized, as we shall illustrate with the visual analysis of image and text collections.

In summary, our method, which we call the Visual SuperTree (VST), is able to visually encode similarity in a multiscale manner, in a visually scalable way, for datasets of hundreds of thousands to millions, at interactive speeds. We believe that this combination of desirable properties is unique to the VST as compared to other existing tree visualization methods. The way in which this is achieved is described next.

3.2 The Visual SuperTree

We next discuss the requirements that a VST should comply of and, in the same time, detail the combination of techniques used to enable interaction, summarization and exploration. An overview of our proposed VST construction pipeline is given in Fig. 3.1.

3.2.1 Construction

The VST requires as input a tree which reflects similarity relationships between observations of a dataset. It also requires that the branches of this tree can be partitioned into several levels of detail, *i.e.*, into groups of highly similar nodes that can be summarized by aggregating them. We call these groups *supernodes*. As such, a VST consists of a similarity tree having as nodes both observations and supernodes (groups of observations). We use the supernode abstraction to allow a consistent and controlled exploration of the data, mainly in the sense of reducing visual clutter when visualizing large datasets. The construction of supernodes is detailed separately in Sec. 3.3 so as not to interrupt the flow of the main narrative, and since the supertree is intimately linked to its visualization methods discussed next.

To build the similarity trees required by VST as input, several methods can be used. We next detail three popular ones:

Neighbor-Joining (NJ): As input, we consider a dataset of n observations $\{x_i\}$, $1 \leqslant i \leqslant n$. From this dataset, we derive a distance matrix $D = (D_{ij})$, where D_{ij} is the distance between observations x_i and x_j in our dataset. This can be done either explicitly, by using the dimensions of the observations to compute distances between them, or the distance matrix can be provided by some other means of characterizing the similarity of observations that does not consider explicit dimensions, as explained in Sec. 3.1 in the context of projections. From D, the NJ algorithm builds an unrooted tree where its n leaves are the observations x_i, and where the $n-2$ non-leaf tree nodes represent 'ancestors' with two children each. NJ starts by creating the n leaves. At each step, the algorithm (i) selects a pair of nodes (i,j) with the smallest sum of branch lengths S_{ij}, defined as the sum of distances of nodes i and j to the lowest level in the tree, expressed as

$$S_{ij} = \frac{1}{2(n-2)} \sum_{k \neq i,j} (D_{ik} + D_{jk}) + \frac{D_{ij}}{2} + \frac{1}{n-2} \sum_{\substack{k<l \\ k,l \neq i,j}} D_{kl}. \tag{3.1}$$

In step (ii), the algorithm adds a node x to the set of current nodes, by connecting x to i and also to j. Next, the algorithm (iii) evaluates the edge lengths L_{ix} and L_{jx} as

$$L_{ix} = \frac{1}{2} \left(D_{ij} + \frac{1}{n-2} \left(\sum_{k \neq j} D_{ik} - \sum_{k \neq i} D_{jk} \right) \right)$$

$$L_{jx} = \frac{1}{2} \left(D_{ij} + \frac{1}{n-2} \left(\sum_{k \neq i} D_{jk} - \sum_{k \neq j} D_{ik} \right) \right). \tag{3.2}$$

Edge lengths L_{ij} model the distance between nodes corresponding to observations x_i and x_j, as captured by the NJ tree. Edge lengths will be next used when drawing the tree, so as to visually reflect the similarity of nodes connected by edges, and thereby construct a final drawing that encodes well the original similarities in D_{ij} that the tree captures.

After computing edge lengths, we (iv) replace rows and columns i and j by a new row (and corresponding column) x in the distance matrix. To do this, it evaluates the distances D_{xy} from the newly added element x to all other elements y in the matrix by

$$D_{xy} = \frac{1}{2} \left(D_{iy} + D_{jy} \right). \tag{3.3}$$

When only three nodes $1, 2, 3$ are left, a final node x connecting them is added, and the lengths of the three corresponding added edges are set to

$$D_{1x} = \frac{D_{21} + D_{31} - D_{32}}{2}, \ D_{2x} = \frac{D_{21} + D_{32} - D_{31}}{2}, \ D_{3x} = \frac{D_{31} + D_{32} - D_{21}}{2}.$$

$$(3.4)$$

After this, the NJ algorithm has completed its work.

WPGMA: The WPGMA (Weighted Pair Group Method with Arithmetic Mean) [170] is a tree construction method based on a bottom-up agglomerative clustering. The algorithm receives a distance matrix D as input, which holds the distances between pairs i, j to all n observations, just like the NJ algorithm. The WPGMA algorithm next creates a rooted tree by grouping the two most similar observations (i, j) each time. These observations will be represented by the nodes n_i and n_j in the tree. Next, a new node $n_{(i \cup j)}$ is created to represent this group, and will become their parent in the tree. For this tree, we also define the edge lengths $L_{(i \cup j), k}, k \in \{i, j\}$ as

$$L_{(i \cup j), k} = \frac{D_{i,k} + D_{j,k}}{2} - S_k, \tag{3.5}$$

where

$$S_{(i \cup j)} = \frac{D_{i,j}}{2} \tag{3.6}$$

represents the total distance of the node k to the lowest level in the tree. Obviously, if k is a leaf node, then $S_k = 0$. The matrix D is then updated to remove the distances for observations i and j and include distances $D_{(i \cup j), k}$ from the new node $n_{(i \cup j)}$ to the remaining k observations. These distances are computed as

$$D_{(i \cup j), k} = \frac{D_{i,k} + D_{j,k}}{2}. \tag{3.7}$$

This process of grouping and updating D is repeated until there is only one node left, which is the tree root. One important aspect of WPGMA is that it creates an ultrametric tree. This means that, given a node k, the total length of its path down to the lowest level of the tree is the same.

UPGMA: The UPGMA (Unweighted Pair Group Method with Arithmetic Mean) [170] is a similar tree construction algorithm to WPGMA. Both are bottom-up agglomerative techniques, but UPGMA uses a different approach to update the distance

matrix. When updated, distances are weighted by the number of nodes contained in each group. That is, UPGMA replaces Eqn. 3.7 from WPGMA by

$$D_{(i \cup j),k} = \frac{\|i\| \, d_{i,k} + \|j\| \, d_{j,k}}{\|i\| \, \|j\|}. \tag{3.8}$$

Best algorithm: All the approaches presented above aim to find the "ideal similarity tree", *i.e.*, a tree which matches the distances in D by the lengths $L_{i,j}$ of branches between corresponding tree nodes. UPGMA and WPGMA are faster to compute ($O(n^2)$ for n observations) than NJ, which is $O(n^3)$. However, the main disadvantage of UPGMA and WPGMA is that they can create misleading results when the ideal similarity tree is not ultrametric.

(a) Test dataset   (b) Neighborhood preservation plots
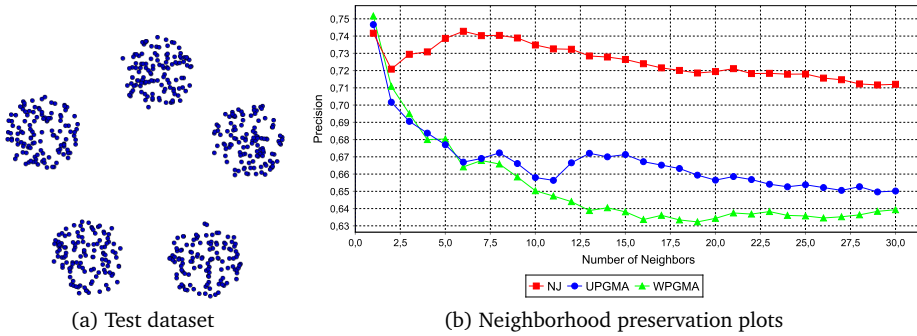
Figure 3.2: Testing neighborhood preservation for three similarity-tree construction algorithms.

The above issues in terms of quality, or distance preservation, of a similarity tree algorithm are very important in our context, since, as explained, we want to visualize such a tree to be able to reason about similarities in the original dataset. To assess quality, we compared the three studied algorithms (NJ, UPGMA, and WPGMA) using a synthetic dataset which contains five well defined separate observation groups (Fig. 3.2a). Here, the distance D corresponds to the pairwise Euclidean distances between the points plotted in the figure. Having this dataset, we ran each algorithm separately, obtaining thus three different similarity trees. Next, we need to assess which of these trees reflects the input data more faithfully. Since we do not have a way to compute the ideal similarity tree to use as ground truth, we proceeded by using a proxy quality metric – the neighborhood preservation (Sec. 2.4.3). In detail, this works as follows: For each observation $\mathbf{x}_i$ in our dataset, we can define a set of k neighbors ν_i^{data} as the k observations more similar to $\mathbf{x}_i$. For the node i in a similarity tree that corresponds to $\mathbf{x}_i$, we define its k neighbors ν_i^{tree} as the k nodes whose sum of edge lengths in their path to i is minimal. The neighborhood preservation π_i is defined as the ratio of observations in ν_i^{data} whose corresponding nodes i are in ν_i^{tree} to the size of ν_i^{data}. For an

entire dataset mapped to a similarity tree, neighborhood preservation $\bar{\pi}$ is defined as the average of the neighborhood preservation π_i for all data items i. Ideally, all values of π_i should be one for any neighborhood size.

Using this model, we evaluated the neighborhood preservations $\bar{\pi}$ for the three studied algorithms by plotting them as functions of neighborhood size, using neighborhood preservation plots (Sec. 2.4.3). The neighborhoods are defined by the k-nearest neighbors, and the values of k used here range from $k = 1$ to $k = 30$ neighbors. Figure 3.2b shows the results. Here, we see that NJ performs significantly better than WPGMA and UPGMA, for roughly all neighborhood sizes. As such, we use NJ in our further tree construction (see also Sec. 3.3). When drawing this conclusion, it is very important to stress that the quality assessed here is that of the neighborhood preservation *of the similarity tree* and not that of a *drawing* of the similarity tree. Indeed, at this point of our design, we want to assess which similarity tree, seen as an abstract data representation, encodes best our initial dataset similarities. Tree drawing, a separate concern, is discussed next.

3.2.2   Tree Layout

Having a similarity tree constructed as discussed above, we need next to draw this tree so that we can use it to assess similarities of our original observations. For this, we use three different tree layout approaches: a radial layout [7], a circular layout [7] and a force-directed layout [25]. Figure 3.3 shows the results for a similarity tree constructed from an artificial dataset wich contains 1000 observations divided in four groups. Details about the construction method used to build this tree are discussed in Sec. 3.2.1. Both radial and circular layouts are specifically designed to work on rooted trees. As root, a central node, *i.e.,* having the smallest sum of shortest paths to all other nodes, is selected. A cheap approximation of the above, which considers all edges of the same length, is to iteratively remove leaf nodes from the tree, in a first-in-first-out order, until only one node is left – the root. Note that this 'visual root' is not necessarily the same node as the root of the similarity tree being constructed as explained in Sec. 3.2.1.

Several observations can be made when we compare the three considered tree layouts, as follows.

Radial layout: This layout is designed to preserve edge lengths (as encoded by the values L_{ij} computed as explained in the previous section) into the visual space as well as possible, and provides a general overview of the tree structure (Fig. 3.3a). The algorithm assigns angular wedges of a circle to the tree branches, so that a wedge's size is proportional to the number of leaves of its branch. Edges are then drawn along wedge-angle bisectors and branches are kept disjoint on the layout plane. This way edges can have any length without violating disjointness. The algorithm works directly with a standard similarity tree. However, as we explained in Sec. 3.2.1, our VSTs can contain both plain nodes and supernodes, the latter

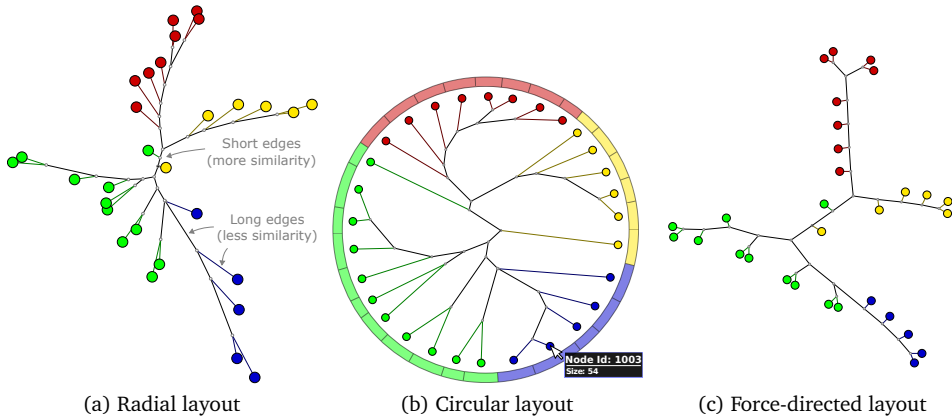(a) Radial layout   (b) Circular layout   (c) Force-directed layout

Figure 3.3: Visual SuperTree layout strategies. The radial layout (a) preserves edge lengths and gives a general view of the tree structure. The circular layout (b) helps comparing the sizes of supernodes in terms of the circular sector sizes. The force-directed layout (c) spreads the tree to better fill the visual space, removing overlaps and allowing a more detailed view of individual branches.

being groups of nodes. To handle this tree structure, we modified the original algorithm [7] to also consider the size of each supernode to define the angular wedges, so the bigger a supernode is, the bigger will be its contribution to define the size of its branch angular wedge.

Circular layout: This layout places leaf nodes equidistantly along the perimeter of a circle. Recall that we have to handle a VST, whose leaves can be either individual observations or supernodes. To do this, we modified the original circular layout algorithm [7] to reserve node wedges proportional to the size of the supernodes, *i.e.*, reserve larger angular wedges to supernodes containing more observations. Figure 3.3b shows a result of this modification. To help the inspection of the wedge sizes, we also display a radial pie chart around the tree, where each circular sector represents the angular wedge reserved for a node. This helps the analysis of cluster sizes and provides an overview of the clustering structure.

Force-directed: This layout simulates a physical system where graph nodes are connected to imaginary springs that iteratively push or pull nodes based on edge weights. Along these forces, nodes repel each other inversely proportionally to the distance between them. The physical system starts with a random positioning of the nodes and iterates the placement until a force equilibrium is reached. We use this approach to spread the tree on the visual space, which enables a more detailed inspection of individual branches. Figure 3.3c shows the result. This layout is also important for the summarization of a VST (detailed next in Sec. 3.2.4), since the

force-directed layout favors an even distribution of free space between nodes and removes most, if not all, node clutter in the drawing.

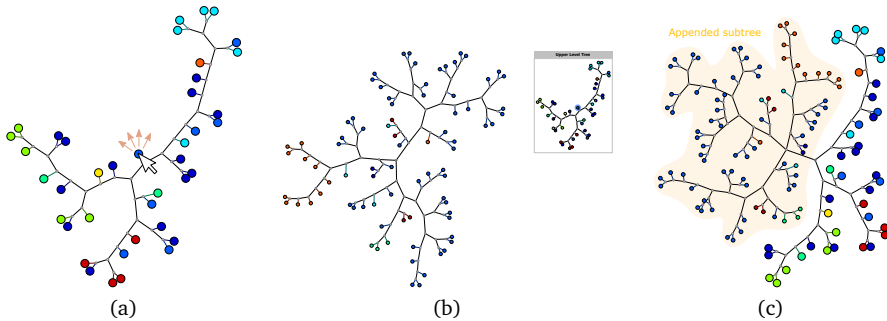### 3.2.3   Multiscale exploration and summarization

Figure 3.4: Multiscale expansion. When a supernode is selected (a), a new subtree can be displayed in a new window (b), or appended into the VST (c). Large nodes represent clusters and small nodes represent data observations.
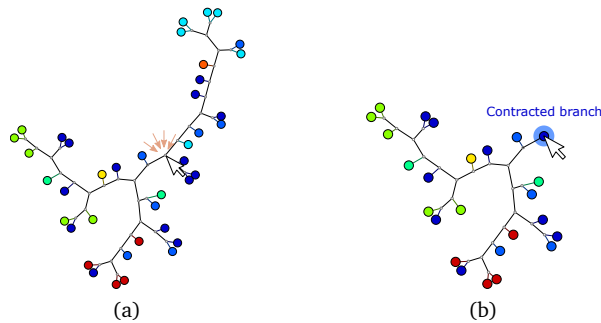

Figure 3.5: Multiscale contraction. Branches can be contracted into supernodes, saving visual space for the remaining nodes (b).

To interact with a VST having millions of observations, we must resort to data aggregation, coordination of the layout, and content summarization. To this end we rely on the similarity-based partitioning provided by clustering techniques.

The primary view of a large VST is typically collapsed, where small nodes represent observations and large nodes represent supernodes, as illustrated in Fig. 3.4a. This favors a fast overview rendering of large datasets. Next, the data can be explored by selecting and collapsing branches, moving nodes, loading contents (*e.g.,* images or text related to nodes), brushing nodes for details on demand, pan and zoom, and other interaction tools, as detailed next.

Starting from a collapsed view (Fig. 3.4a), selecting a supernode triggers the display of the NJ tree for that cluster, as illustrated in Fig. 3.4b. When a supernode is expanded, an overview of the previous coarse view is shown in a mini-map that also highlights the entry point to the current level of detail, *i.e.*, the expanded supernode. This allows the user to keep track of the navigation, or maintain the mental map, and also to return to the previous level of detail through the mini-map. It is also possible to append the finer-grained level tree into the coarse-level tree visualization, as shown in Fig. 3.4c). This approach allows creating views of the entire dataset where different parts have different levels of detail in the visualization. On the other hand, the view can become cluttered when many fine-level trees are appended together – in the limit, such a view is equivalent to displaying the raw similarity tree, when all supernodes have been expanded. Hence, one should keep a balance during the visual exploration of the VST between appending fine-grained views to the overview or displaying them separately in a new window.

The reverse procedure to expanding supernodes is contracting tree branches into supernodes. This way, the selected branch will be displayed on a coarser scale, by being replaced by a supernode in the current visualization. Figure 3.5b illustrates this. Contracting branches allows a selective selective visualization of the VST by saving visual space to allocate to the remaining expanded branches present in the current level of detail. This feature is especially important when exploring large datasets through visual summaries, as detailed next in Sec. 3.2.4.

3.2.4   Data summarization

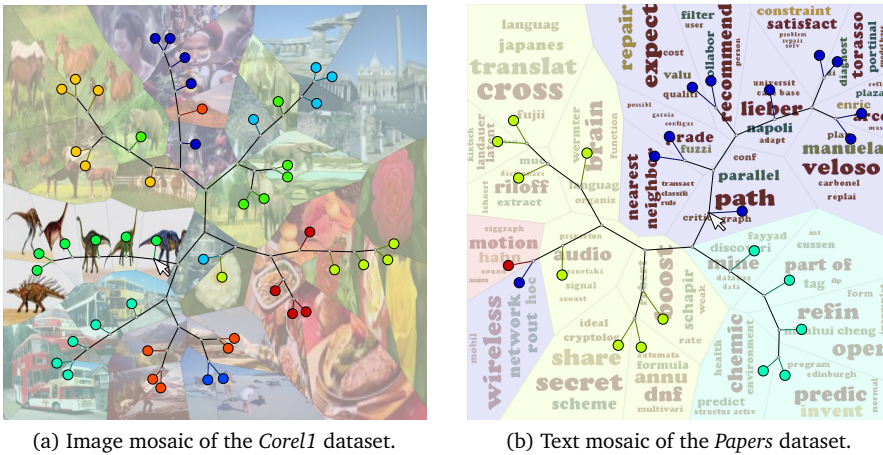(a) Image mosaic of the *Corel1* dataset.   (b) Text mosaic of the *Papers* dataset.

Figure 3.6: Summarization of VSTs exploring image and text datasets. We emphasize selected branches by increasing mosaic cell opacities.

Summarization is a key feature when exploring large datasets. VSTs are amenable to summarization because they group observations along branches by similarity, and because trees admit a planar overlap-free embedding. Indeed, the first factor (similarity) ensures that computing relevant summaries is easy, as the elements to summarize are similar. The second factor (planar overlap-free embedding) ensures that we can display visual summaries without running the risk that these would overlap, which would render them hardly readable.

Similarly to the approach adopted by ImageHive [180], we divide the 2D visual space around tree nodes through a Voronoi diagram. Thus, each data observation or cluster represented by a node becomes associated with a cell of the diagram. Next, each cell may be filled with a texture that represents the data, like images or text (see Fig. 3.6). Details on the image and text datasets used here are given further in Sec. 3.3.1, Tab. 3.1. We call this Voronoi-based summarization a *mosaic*. To communicate the data structure, we overlay the tree drawing atop of this mosaic.

Text mosaics are built by employing a topic extraction algorithm based on the relevance of document words followed by a polygon filling algorithm [143]. This algorithm arranges the most important terms (topics) so as to fit in and fill the convex polygons representing Voronoi cells, whereby the text for the terms are scaled to reflect their relative importance in the summarization. When text is to be shown for a plain observation (single text document), terms are extracted from this document only. When we treat a supernode (collection of documents), terms are extracted from the union of all documents corresponding to observations in the supernode. For images, we proceed differently: When an image is to be shown for a plain observation (single image), we naturally use that respective image. When we treat a supernode (collection of images), we use the supernode's cluster medoid to retrieve the associated image to be displayed.

The user can focus on part of the data by selecting a tree branch. This highlights the related cells. This operation conceptually supports the query "show me all data items, or summarizations thereof, that correspond to the concept being aggregated by the node at the root of this branch". Selecting more central nodes thus shows larger portions of the data space. Selecting nodes closer to the tree periphery (leaves) selects smaller, and more specific, portions of the data space. It is also possible to coarsen the mosaic level of detail, by joining neighboring cells in a new cell. In this case, a new texture (text or image) has to be constructed for the new cell. This is done by defining a distance threshold in the visual space. Next, all tree leaves that are at a distance smaller than this threshold from the indicated location where the join should occur, are grouped in a supernode. Finally, this supernode is summarized as indicated earlier for the original VST.

When a cell represents a group of observations, *i.e.*, a supernode, scrolling down from the top of the cell shows the sequence of images associated with that region. This is similar to the so-called 'flicking' interaction present in many image gallery tools. It is also possible to move, separate, and merge nodes and branches to readjust the diagram.

3.2.5 Alternative explorations

A typical issue during the exploration of moderate to large-size datasets is find-
ing out the optimal number of underlying groups that helps unveiling unknown
relations in data. As a bottomline, this can be done by manually expanding or
collapsing various supernodes of a given VST, until an optimal local level-of-detail
is obtained. However, this procedure requires significant user effort.

A different approach is to use several clustering algorithms to produce a family
of VSTs from the same input dataset. From these, the user may select one or sev-
eral for more for in-depth exploration. The same idea may be applied recursively
to subsets of the data, *i.e.*, to subtrees of the VST. To overcome the potentially
high computational cost of computing many different clusterings, a dimensional-
ity reduction technique can be used to reduce the data dimensionality to a few
dimensions that capture the essence of the similarities of the underlying obser-
vations. Next, this reduced dimension set can be used for clustering. This makes
clustering algorithms faster, since computing similarities is now faster. However,
this trades off precision, as the reduced set of dimensions typically only approxi-
mates the original distances, as discussed in Sec. 2.4.3.

3.3 VST Construction Example: Multiscale NJ

As outlined in Sec. 3.2.1, similarity trees are too costly to display, and also gen-
erate too much clutter, to serve for the visual exploration of large datasets. As
such, in that section we have introduced the idea of a supertree that consists of
observations and supernodes (groups of observations). In this section we explain
how to construct a supertree that has the multiscale properties we require for a
fast and effective exploration of large datasets.

ALGORITHM: As described in Sec. 3.2.1 too, the Neighbor-Joining (NJ) algo-
rithm offers a good solution to the construction of similarity trees due to its preci-
sion. Therefore we selected this algorithm as an basis to build a VST. To overcome
the computational limitations of the NJ technique, we construct the VST by par-
titioning the input dataset recursively. This procedure is guided by the parameter
s_{max} which defines the maximum cluster size in terms of observations. If a clus-
ter is larger than s_{max}, it is recursively subdivided, thereby defining a hierarchy
of clusters. The sizes of the smaller clusters emerging from this subdivision are
evaluated, and this procedure continues until there are no clusters larger than
s_{max}. At this point, an NJ tree is constructed directly for each cluster, and the re-
sulting NJ trees are connected by other NJ trees between partition levels. A high
level pseudo-code for our clustering algorithm is given in Algorithm 1. In here,
$clustering()$ can be any clustering algorithm that delivers a partition of the in-
put dataset into several clusters. This can be either a single-level method, such as
k-means, or a multilevel method, such as hierarchical bottom-up agglomerative

clustering (Sec. 2.2). In case of multilevel methods, we do not use the produced hierarchy, but simply select a single, typically coarse, level. Indeed, the multilevel data partitioning is created by our own algorithm, the top-down data-division procedure *SuperTreeConstruction()*.

Algorithm 1 Algorithm for the VST construction.

1: **function** SUPERTREECONSTRUCTION($data, s_{max}$)
2:     **if** $data.size < s_{max}$ **then**
3:         matrix $D \leftarrow$ evaluate-distances($data$)
4:         **return** NJ-tree(D)
5:     **else**
6:         $C \leftarrow$ clustering($data$)
7:         Let $\overline{C}$ be a new matrix
8:         **for all** cluster $c_i \in C$ **do**
9:             $c_i.tree \leftarrow$ SuperTree($c_i.data, s_{max}$)
10:        add $c_i.centroid$ to $\overline{C}$
11:         **end for**
12:         matrix $D \leftarrow$ evaluate-distances($\overline{C}$)
13:         **return** NJ-tree(D)
14:     **end if**
15: **end function**

Any clustering algorithm that partitions data according to similarities between observations can be used as basis for a VST construction. Of course, different such algorithms will yield different data hierarchies and different construction times. The value of s_{max} will also impact the same issues, since it defines the size of data chunks that are managed by the tree construction algorithm. We address these questions in Section 3.3.1, where some experimental results are shown for real and artificial datasets.

The Neighbor-Joining (NJ) algorithm [157] is used to connect observations in the lowest tree level and clusters in the other levels, respectively. Figure 3.7 shows two views of supertrees for a collection with 12000 instances from 4 classes. Class values – a categorical attribute – are mapped to colors, but are not used in the clustering process. This visualization serves as a simple visual means to ascertain whether similar entities have been indeed grouped in the same subtrees. To construct the VST we used here the k-means clustering algorithm, with $k = 10$ clusters and $s_{max} = 100$. The figure shows a collapsed version of the tree (left) and an fully expanded version of the same tree (right), along the lines of our proposed multiscale visual exploration discussed in Sec. 3.2.3.

SCALABILITY IMPROVEMENTS: We also implemented a strategy for building VSTs and storing them on disk prior to visualization and exploration. This strategy has the advantage of allowing the computation of clusters and VSTs for larger datasets offline using parallel clustering approaches. In turn, this enables a quick
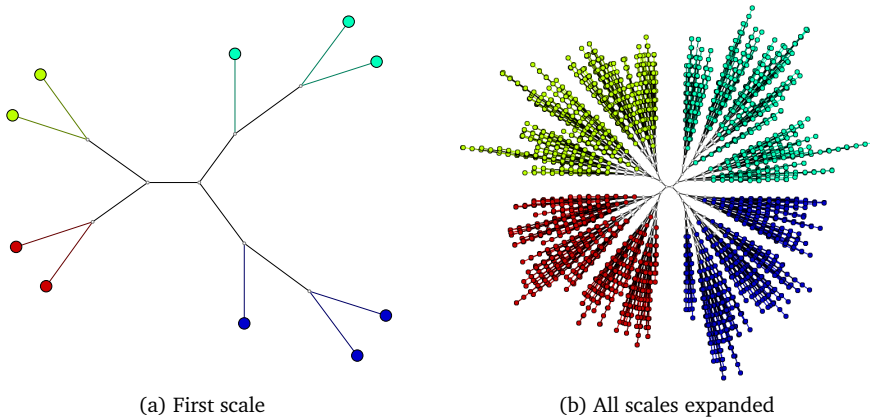
(a) First scale   (b) All scales expanded

Figure 3.7: Example of a VST for a collection of $12,000$ observations. Instance classes are mapped to color for easy assessment of similarity.

exploration of a large VST on commodity computers by loading the required information on demand, without overloading the main memory and without having to re-run the expensive clustering processes. We use binary files to store records sequentially. Each record holds the configuration of one VST level, *i.e.*, which clusters are computed for the level, the medoids for each cluster, labels related to each medoid, and data summaries per cluster, such as number of elements per cluster. The record also holds the tree topology and edge lengths in the Newick format [144]. Indexes are used to store the position of each record in a data file. Records are loaded on demand as specific VST levels are explored. In this way, the original (large) multidimensional data is used only to compute the clusters and the VST levels, and is no longer required during the actual interactive exploration. All in all, these mechanisms lead to smooth interactive exploration and visualization on commodity computers having limited amounts of RAM, even for large datasets.

3.3.1 Experimental evaluation

Point placement, either by multidimensional projection or by similarity trees, have good precision in terms of preserving neighborhoods from the original space. As datasets grow larger and have more dimensions, however, it is harder to achieve high precision with any technique. Clustering-based partitioning, such as the one described above, might further compromise precision, especially when it is a greedy method like the one we propose. To assess the quality of VSTs in terms of similarity-based point placement and of computational scalability, we performed a series of experiments. These experiments led us to conclude that VSTs are very

fast to explore and fairly preserve neighborhoods of observations even for large datasets. These experiments are described next.

Table 3.1: Datasets used in our experiments.

Dataset	Description Type	Size	Attributes	Classes
Papers	VSM	675	1423	4
Corel1	SIFT	1,000	128	10
Corel2	BIC	3,906	128	85
Freephoto	BIC	3,462	128	9
Caltech	CCV	9,144	128	101
RBF4K	RBF	4,000	4	4
RBF12K	RBF	12,000	4	4
RBF20K	RBF	20,000	4	4
Imagenet Fall 2011	SIFT	1,261,406	1000	-
Stackoverflow	VSM	1,056,010	1693	-
KDD	intrusion attributes	4,898,431	38	5

DATASETS: Table 3.1 presents the datasets that were used in the various experiments and examples reported in this chapter. *Corel1* and *Corel2* are photographs (images) represented by scale-invariant features (SIFT) [113] and border-interior pixel classifications (BIC) [174]. These are well-known descriptors used to represent images as multidimensional feature vectors in applications such as image classification and retrieval. The *Papers* dataset contains extracts (title, authors, abstract and references) of scientific papers from 4 research areas processed by the conventional vector space model (VSM). *Caltech* [59] contains images represented by color coherence vectors (CCV) [138], another standard way to represent images as multidimensional data. *RBF4K*, *RBF12K*, and *RBF20K* are synthetic datasets, used as ground truth data to understand how VSTs behave. They all contain four well-separated, dense, hyperspheres covered by dense point sampling, and were generated by using radial basis functions (RBFs) using the MOA toolkit [14]. *KDD* is a popular and widely used dataset considered for the KDD Competition, consisting of network intrusion reports data [6]. *Freephoto* contains nature photos selected at random from *www.freefoto.com* and next represented by BIC descriptors. *Imagenet Fall 2011* [41] is a dataset sampled from a large ontology of images organized in a hierarchical structure. It is represented as a bag-of-visual-features using SIFT descriptors. The *Stackoverflow* dataset is a subset of textual documents from the MSR Mining Challenge 2015 [212]. It contains discussions between users of the Stackoverflow community from August, 2008 to September, 2014, represented using the VSM model.

As visible from the above overview, the considered datasets cover a wide variety of application domains, data types, sizes, dimensionalities, and number of

class labels. What is common to all, is that the dimensions aim to capture similarity in the data. As such, we believe this collection is a good test dataset for the performance of VSTs.

CLUSTERING: Clustering algorithms and their evaluation have already been addressed extensively in the literatire [93]. We chose k-means [92] as clustering algorithm in our experiments due to its speed, precision, ease of use, implementation simplicity, and popularity. For this technique, the user has to define the number of clusters to partition the data, which is one of the major drawbacks of the algorithm, since the best number of groups is seldom known beforehand. As a guiding estimate for k, we use use $\sqrt{N}$, where N is the number of observations in the dataset. This is a frequently accepted as an upper bound for the actual number of clusters to be found in the data [137].

The value for the parameter s_{max} directly impacts tree construction. Larger values will trigger the clustering algorithm less frequently, which means less computation time, specially for large datasets. However, larger values will also cause the construction of larger leaf nodes, which might be more expensive to further process due to the cost of the single-level NJ algorithm (Sec. 3.2.1), which is cubic in the number of input data points.

Table 3.2: Construction times for VSTs and NJ trees, in seconds.

| Dataset	Clustering	s_{max}	VST	NJ Tree
RBF4K	4-means	100	6.04	65.82
	4-means	250	3.76	
	4-means	500	3.04	
	4-means	1000	3.55	
RBF12K	4-means	300	16.34	1671.10
	4-means	750	13.84	
	4-means	1500	18.43	
	4-means	3000	48.46	
RBF20K	4-means	500	41.01	–
	4-means	1250	40.41	
	4-means	2500	64.57	
	4-means	5000	201.27	
Imagenet	200-means	500	337,515.88	–
KDD	150-means	250	20,976.30	–
	150-means	500	20,384.99	
	150-means	1000	20,223.74	
Stackoverflow	100-means	300	52,782.91	–
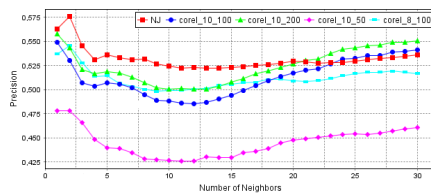
(a) *Corel1* dataset. Clustering: 8-means (s_{max} = 100, cyan) and 10-means (s_{max} = 50 (pink), 100 (blue), and 200 (green)). Plain NJ algorithm in red.
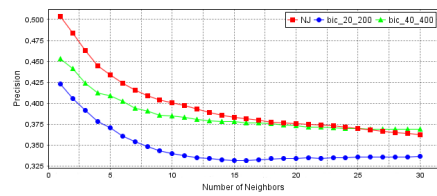
(b) *Corel2* dataset. Clustering: 20-means (s_{max} = 200, blue) and 40-means (s_{max} = 400, green). Plain NJ algorithm in red.
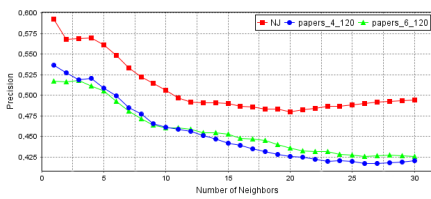
(c) *Papers* dataset. Clustering: 4-means (s_{max} = 120, blue) and 6-means (s_{max} = 120, green). Plain NJ algorithm in red.
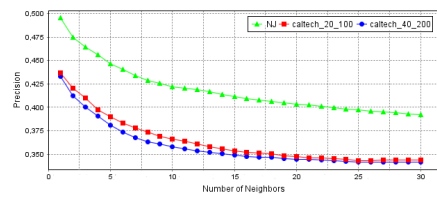
(d) *Caltech* dataset. Clustering: 20-means (s_{max} = 100, red) and 40-means (s_{max} = 200, blue). Plain NJ algorithm in green.

Figure 3.8: Neighborhood preservation (2 to 30 neighbors) for different datasets and clustering parameters.

TIMINGS: We considered different values of s_{max} in our experiments. These, and the resulting times, are outlined in Tab. 3.2. For the smaller datasets *RBF4K*, *RBF12K*, and *RBF20K*, computations use a laptop with Intel i7 processor at 2.50GHz and 4GB RAM. The times for building a plain NJ tree for the whole data set are also shown for the smallest two datasets, *RBF4K*, *RBF12K*. For the other datasets, we did not compute plain NJ trees, since the cubic computational complexity and square memory complexity makes such computations far too expensive. It is clear that building a VST is much faster than building an NJ tree. The table also shows the relation between the time spent by the clustering algorithm and the time spent by NJ under influence of s_{max}. The larger *Imagenet*, *KDD*, and *Stackoverflow* datasets were processed on a desktop PC with Intel Xeon E5-2630 processor at 2.3 GHz and 64GB RAM. For these datasets, we stored the VSTs on disk for later exploration. While VST computation trees are, in absolute value, very large for these datasets, we stress that plain NJ trees would be simply impracticable, due to the aforementioned much higher computational and memory costs.

QUALITY: Neighborhood preservation is a means to assess layout precision through neighborhood relationships, as discussed in Section 3.2.1. Figure 3.8 presents a plot of neighborhood preservation values for the range of 2 to 30 neighbors for the *Corel1*, *Corel2*, *Papers*, and *Caltech* datasets. We can see that, although precision for the VST is generally smaller when compared to plain NJ trees, its behavior as a function of the neighborhood size k closely resembles that of NJ trees.

As expected, starting k-means with k closer to the actual number of classes typically improved precision. The effect of s_{max} can also be observed in Figure 3.8a: Larger values allow direct construction of larger NJ trees at lowest levels, improving precision. Separately, the plot for the *Papers* dataset shows that a large number of dimensions (1423 in this case, see Tab. 3.1 did not cause a larger drop in relative precision as compared to the plain NJ algorithm and as compared to other datasets. Overall, this evaluation shows that VSTs are computationally far more scalable than classical NJ trees, and offer an arguably good trade-off between being able to handle larger datasets but offering lower precision for neighborhood preservation.

3.4   Example Applications

As stated several times so far, the main goal of VSTs is to allow the visual exploration of large datasets based on their organization by similarity. Additionally, the VST approach enables useful explorative interactions when points or groups of points can be summarized meaningfully, as it is the case for images and text. We now exemplify the exploration approach we implemented atop of the VST visualization, as well as its functionality for exploration of image and text collections. The exploration is done by a tool in which we have implemented the VST compu-

tation, display, summarization, and interactive navigation mechanisms described so far.

3.4.1   Exploration of network monitoring data

Figure 3.9 shows snapshots of the exploration of the *KDD* dataset, a collection of network connections composed by $4,898,431$ observations divided in 5 classes (see also Tab. 3.1). Observations model connections in a computer network. Class values identify the type of each connection, which can be regular or attacking. The attacking class is further specialized into *probe*, denial of service (*dos*), user-to-root (*u2r*), and remote-to-local (*r2l*). Throughout the application we map the class attribute to a categorical color scheme: dark blue for regular connections, light blue for *probe* attacks, green for *dos* attacks, yellow for *u2r* attacks, and red for *r2l* attacks. When visualizing supernodes in the tree, we assign to them the class color corresponding to the cluster medoid. Since class values are categorical attributes, this is equivalent to the class that is dominant, in number of observations, in the respective cluster. While this does not reflect the distribution of summarized class types, it is a simple and easy to understand mechanism. It can be easily enhanced to show more information, such as *e.g.* the percentage of summarized nodes that corresponds to the maximal class, encoded for instance in the color saturation. Other schemes are possible to show more information on the summarized nodes. We leave such refinements for future work.

(a) Radial layout.   (b) Circular layout.
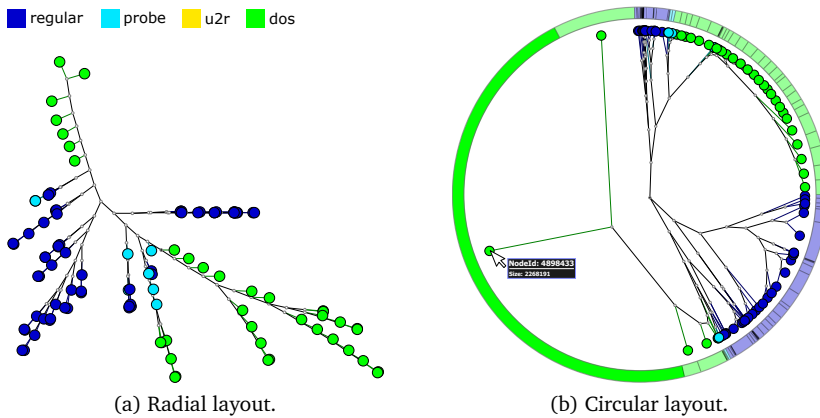
Figure 3.9: Coarsest scale of a VST for the *KDD* dataset.

To generate the VST we have chosen $k = 150$ clusters and $s_{max} = 500$. We selected s_{max} based on experimenting with the visual screen occupancy by the resulting VSTs, so as to have not too many nodes displayed at finer scales, but also have a not too deep tree. On the coarsest scale, (Figure 3.9a), we see two large regions relating *dos* attacks (dark blue) and regular connections (green).

A small number of supernodes were assigned to *probe* attacks (cyan). Using the circular layout, we observe a high unbalanced clustering result, where *dos* attacks dominate the first scale (Figure 3.9b). Here, a single cluster, indicated by the mouse location, contains more than 2 million observations. This is expected due to the nature of *dos* attacks: These consist of attacking servers that flood the network with several connections from various sources, so are typically high-volume in the number of connections.

(a) Scale 2, *dos* attacks.   (b) Scale 3, *dos* attacks.   (c) Scale 3, *probe* attacks.
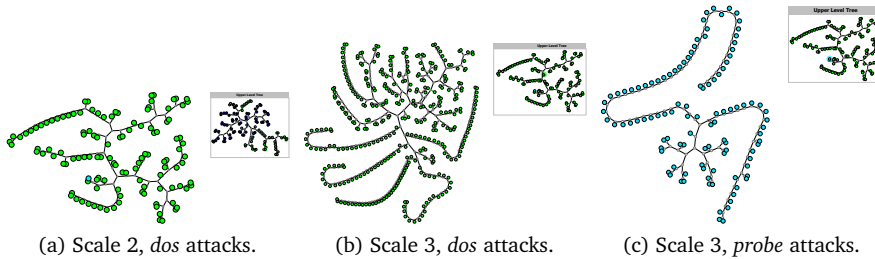
Figure 3.10: Exploration of a $4,898,431$ data elements tree. In this case $s_{max} = 500$ and 150-means. Three levels of exploration on the tree's branches are shown.

To illustrate the exploration of a supernode in more detail, a *dos* supernode was next selected and expanded for investigation (Figure 3.10a). At this second level of detail, we notice long tree branches, indicating high similarity among clusters in the same branch. In other words, there are quite large sets of *dos* attacks that behave similarly. If we expand next a cluster from this level, we obtain, on the third scale, the leaf nodes or actual observations (Figure 3.10b). We see here, even better than on the second level of detail, the long-branch structure of the tree. To get more insight into the meaning of these branches, we inspected the multidimensional features of their nodes, and discovered that these are identical for nodes along a branch. The same happens for other classes (Figure 3.10c). This indicates many repetitive observations (connections for an attack) in this dataset.

3.4.2 Exploration of large image collections

The VST is capable of representing any dataset for which similarity may be defined. As such, image visualization based on similarity is a natural target application for our technique. From previous work on visualizing image collections, we extract a number of desirable requirements, such as showing the similarity of images, providing overview, and preserving structure in terms of groups of highly similar images [128]. These requirements are also the aim of our VST. We note in this context that some other visualization approaches for image collections also offer semantic views [203]; however, the multiscale aspect as well as drilling-down strategies of such approaches are limited.

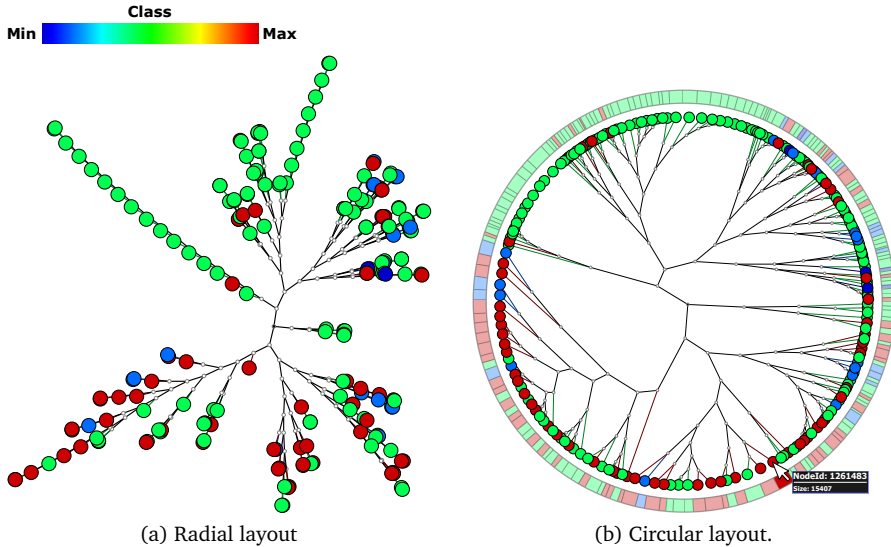(a) Radial layout   (b) Circular layout.

Figure 3.11: First scale exploration of the Imagenet dataset. The radial layout indicates the group separation, and the circular layout indicate the clustering balance.

Figure 3.11 shows part of the exploration possibilities offered by a VST for image collections. Here, we explore the *Imagenet Fall 2011* dataset containing $1,261,406$ observations (see also Tab. 3.1). This dataset is organized in a hierarchy, where the first level contains 9 image classes: plants, geological formation, natural objects, sport images, artifacts, fungi, persons, animals, and miscellaneous. We used these image classes as categorical attributes in our dataset. To construct the VST, we selected 200 clusters and set $s_{max} = 500$.

Using the radial layout (Figure 3.11a), it is possible to detect well defined groups of branches, indicating structure in terms of different similarities in the data. However, we also see that image classes are mixed among a branch, especially once we get closer to the root. Note that this does not necessarily indicate that our VST erroneously places images which are different in terms of their attributes (SIFT descriptors) closely to each other. Indeed, this layout can also indicate that the used SIFT descriptors are not able to provide a good discrimination between the nine high-level class types used as annotations in this dataset.

Further on, we used the circular layout (Figure 3.11b) and noticed that k-means forms a balanced data partition. Finally, we used the force-directed tree layout to create a visual summary of the dataset (Figure 3.12a). Here, space around the tree nodes is partitioned into Voronoi cells and textured by representative images in a cell's cluster, as explained in Sec. 3.2.4. Here, we see some challenges of the proposed mosaic construction: Cell shapes and aspect ratios, can vary significantly, making it hard to fit rectangular images in them well. Moreover, peripheral nodes, which typically correspond to leaves, get more space than central nodes,

which correspond to higher-level concepts describing larger groups of images. As such, the summarization by images dedicates more space to the former than to the latter. Possible improvements of the mosaic, using *e.g.* Voronoi-diagram modifications such as proposed by GMap (Sec. 2.3.2), can be considered in future work to alleviate such issues.

However, the current VST design allows alleviating the above issues by different mechanisms too. For this, we select some branches, based on their density and length, to contract into supernodes. The result, showing the contracted nodes in white, is illustrated in Fig. 3.12b. This mosaic contains larger cells, which are easier to visualize than the original mosaic in Fig. 3.12a. We next decided to drill down in a cell (supernode) to explore it in more detail. To simplify the mosaic at this second scale, we also decided to merge cells corresponding to leaf nodes which are at a small distance from each other in the tree. The resulting visualization shows a mix of images related to sea landscapes, animals, and miscellaneous images (Figure 3.12c). We next repeated the process of drilling down, by expanding a cell in this level-2 mosaic, and reached the third and deepest level of the VST (Figure 3.12d). The mosaic for this level displays a more specialized (similar) set of images, most related to sea landscapes and animals.

3.4.3   Exploring text collections

In our second application, we illustrate the capabilities of VSTs for the exploration of textual datasets. For this, we analyzed documents from the StackOverflow portal, a knowledge-sharing community of programmers, focusing on the exploration of the main topics of this portal[1]. This community, arguably one of the largest and most prominent of its class, is to let programmers ask and answer questions related to software development in general. Users can also mark questions with specific words (tags) to facilitate the search for topics of interest. Our dataset contains programmer discussions ranging from August, 2008 to September, 2014. From this period, we extracted 20,000 documents per month. Due to the unbalanced number of messages per month, this resulted in in 1,056,010 messages in total. The month is also used as an attribute of each message.

To characterize messages in terms of similarity, we use the vector space model (VSM), which is a typical way to treat text data, as already outlined. For this, we first manually selected a set of 1693 keywords from the most frequent message tags. Next, we parsed the messages' contents and built a document-term frequency matrix for the selected set of keywords. We next used this matrix to generate and store the VST on disk, as described in Sec. 3.2.1. In this process, we used k-means with $k = 100$ clusters and $s_{max} = 300$ so as to keep a good balance between processing time and space occupation on screen for each level of the VST.

The resulting VST is shown at its coarsest scale in Figure 3.13. The radial and circular layouts (Figures 3.13a,b) show how groups are separated. We also show
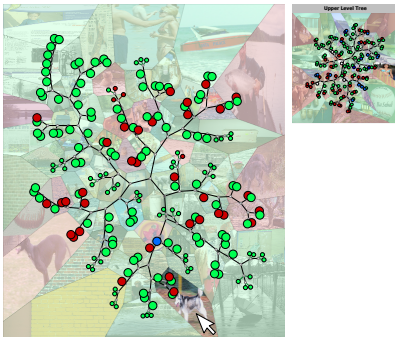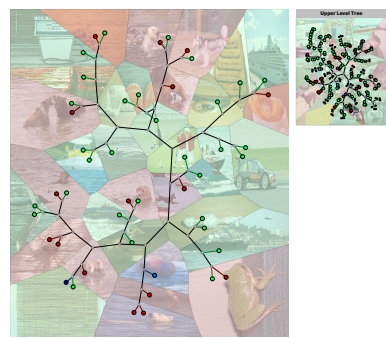
---

1 http://www.stackoverflow.com
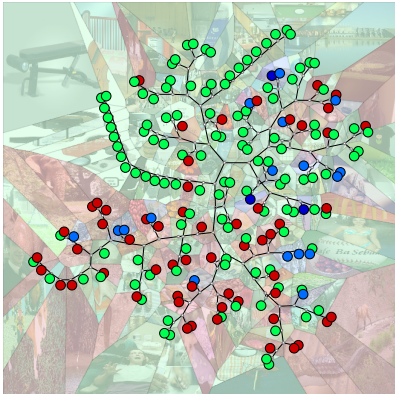
(a) Force-directed layout and mosaic of the coarsest scale.
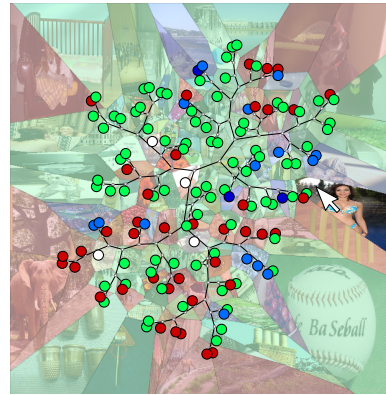
(b) Simplified tree and mosaic of the coarsest scale.

(c) Scale 2.

(d) Scale 3.

Figure 3.12: Exploration of the *Imagenet Fall 2011* dataset, with $k = 200$ clusters and $s_{max} = 500$. Contracting branches into supernodes and merging cells helps to simplify the image mosaic.

the nodes' messages' dates encoded on a dark brown-to-light yellow ordinal colormap. Next, we used the force-directed layout as input to create a text mosaic (Figure 3.13c). The text mosaic summarizes the VST by showing an overview of the main topics in the dataset, as described in Sec. 3.2.4. In the mosaic we notice some branches on the left with topics related to web development, while branches on the right indicate database discussions. The top-left part of the mosaic summarizes discussions related to Windows, Java, Apple and Android development. In the remaining mosaic areas, we can notice a mix of different subjects, like data structures, compilation strategies and programming languages.

Let us next illustrate a targeted subject-based exploration. If we want to search for discussions related to Apple technologies, we can select the cell of this topic and expand it to explore a lower level of the tree. This lower level (Figure 3.13d)

(a) Radial layout

(b) Circular layout

(c) First level mosaic

(d) Apple level mosaic

(e) iPhone/iPad level

(f) Selected branch content
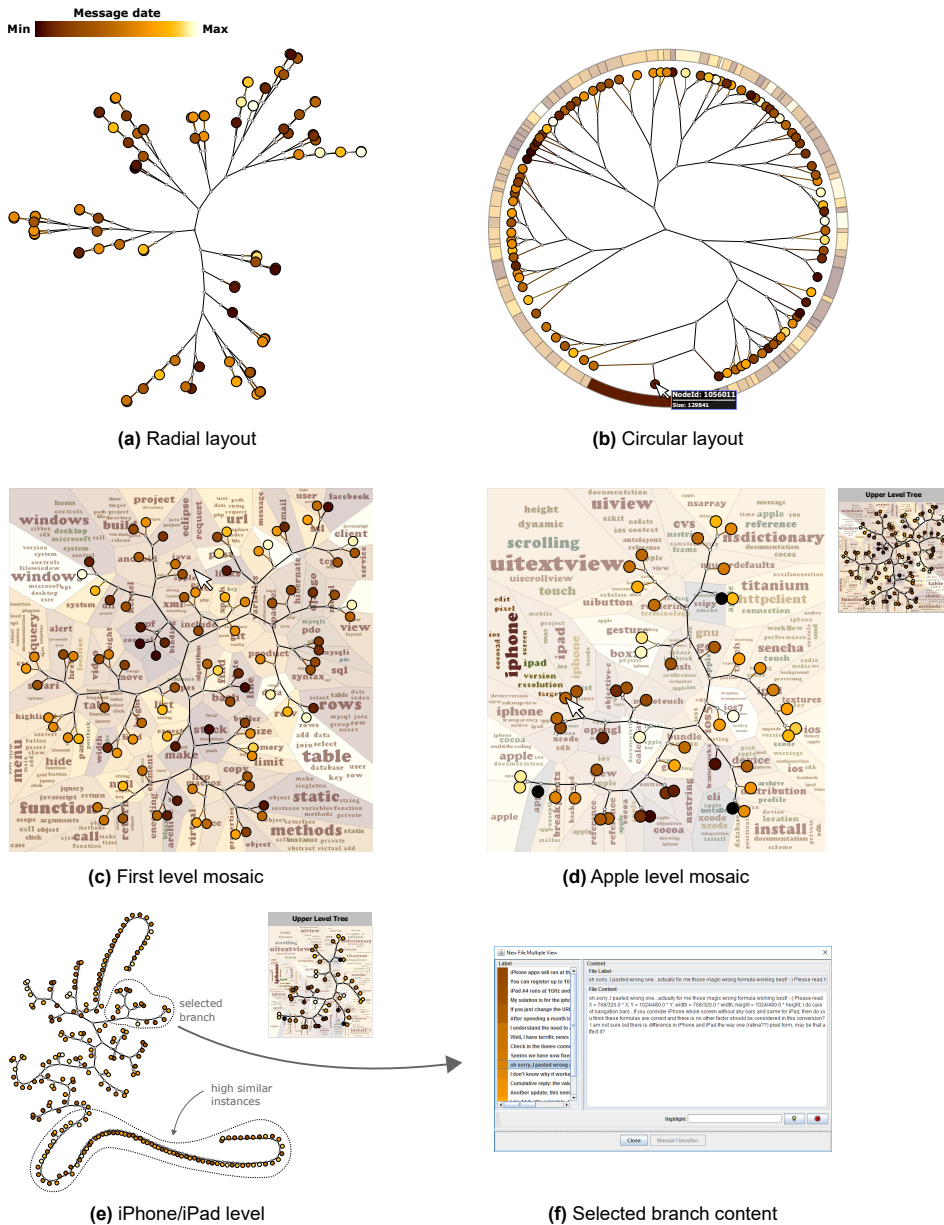
Figure 3.13: Visualization of the Stackoverflow dataset. The radial (a) and circular (b) layouts indicate group separation and cluster balance, respectively. A text mosaic is created for the force-based layout on the first scale to summarize it, showing the its main topics (c). Expanding the subtree from Apple enables exploring specific groups of messages in more detail (d). The deepest level of the iPad/iPhone group can be explored by selecting branches (e) and displaying the content of individual messages (f).

contains groups of topics regarding Apple technologies, such iPhone, iPad, iOs and xCode. Let us say that we want to explore topics related to mobile Apple hardware. For this, we select and expand the iPhone/iPad cell, and we reach the deepest level of this branch (Figure 3.13e). This level which contains 232 messages. We can then select branches of this tree to inspect the content of individual discussions in detail. For example, when selecting the region highlighted on the tree, we found a group of discussions about iPad resolution screen issues (Figure 3.13f).

This particular subtree shows two very long branches (top and bottom parts of the image), which indicate many documents with very similar content. After inspecting them by brushing, we noticed that they represent very short messages about iPad and iPhone issues. They are grouped on a single branch only due to the presence of the words 'iPad' and 'iPhone' on their vector representations, being thus considered identical in the similarity computation procedure.

Overall, the grouping proposed by the VST for this text dataset succeeds in clustering similar items (messages) better than the grouping of images in our first application (Sec. 3.4.2). This is not surprising, as the VST model used here is known to capture quite well the similarity of short and specific text fragments, such as technical messages in an IT internet forum are. Moreover, a careful manual selection of the keywords that drive the features' construction, as we have done here, can massively improve the quality of the induced distance function, since this actually 'samples' the semantic space as the user desires it. In contrast, as already mentioned in Sec. 3.4.2, automatically extracted SIFT features are, in general, too low-level to compute a semantically accurate distance function over large collections of images having a high variability. As such, and since the VST visualization gives good results for exploring the documents dataset, we conjecture that it can be effective for exploring other datasets of other types too, as long as the available dimensions accurately capture the notion of similarity specific to the application domain.

3.4.4   Using VSTs to understand class structure

As a final example of the use of VSTs to understand multidimensional data, we show how VSTs can help understanding the behavior of labeled image data. Here, we proceed inversely than for the example in Sec. 3.4.2: Rather than clustering by image features, we cluster the first level of the tree by class values. Doing this for the *Caltech* dataset yields 101 clusters, since this dataset has 101 different class values. For the deeper levels of the tree, we use the image features to construct the VST, as in the earlier examples. This essentially structures each class of images in terms of similarity measured by actual image attributes.

The first level (Figure 3.14a) shows the coarsest level of the VST. Next, we can select a supernode, such as the one showing flowers indicated in the figure. This unveils a finer-level tree, which shows how flowers are organized in terms of similarities (Figure 3.14b). This example shows that VSTs can be consstructed

by hybrid clustering procedures, where different types of attributes are used per level. In practice, when one has annotated datasets, or reliable ways to extract semantic-related attributes from the data, using such attributes to construct the coarsest level(s) of the tree is preferred. The lower levels of the tree can be constructed by using attributes having weaker semantics which are extracted from the dataset itself.
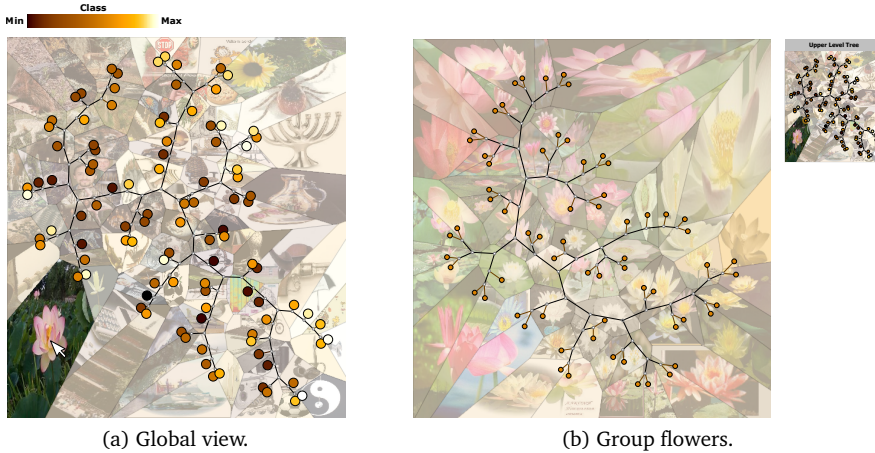

(a) Global view.   (b) Group flowers.

Figure 3.14: Exploration of the Caltech data set by labels. First level has one representative per label. Expanding the flowers group shows its internal similarity structure.

3.5   Discussion

The most important property of the Visual SuperTree is its capability to preserve, up to large extents, the ability of earlier similarity trees (such as the NJ tree) to encode a similarity matrix, while being considerably faster to construct, using less memory for the construction, and offering a multiscale perspective on the data, which makes exploring large datasets faster and less cluttered. The examples shown in this chapter have outlined that the VST is able to handle generic multidimensional datasets having millions of observations and hundreds of dimensions.

The multiscale aspect is present in two senses in the VST. First, the construction of the VST uses a hierarchical top-down clustering method to split the data recursively into increasingly similar, and smaller, clusters. This implicitly makes the execution of the original NJ tree algorithm faster. The depth and breadth of the clustering tree can be controlled by the user by setting, in our implementation, the s_{max} and k parameters of the algorithm. However, any other clustering algorithm that produces a multilevel representation based on the underlying data similarity can be used as well. Secondly, the VST uses this tree to display the data organized along similarity in a hierarchical fashion. This allows visualizing similarities at

various levels of detail, *e.g.*, between entities or groups of entities. Interactive exploration mechanisms allow users to simplify (or refine) the visual representation either uniformly, *i.e.*, one level at a time, or locally, *i.e.*, by expanding or collapsing specific parts of the tree. This allows effectively re-organizing the view so as to bring into focus those (groups) of entities which are best suited to understand the data structure.

In terms of visual mapping, the VST is displayed by a mix of relational visualization techniques (tree layouts), space partitioning techniques (Voronoi diagrams), and annotations (images, tag clouds, and color mapping). These allow not just summarization, but also a display which is both observation-centric (as it shows the observations or groups thereof as points) and dimension-centric (as it shows values of the salient dimensions that characterize a group or observation).

Alternative mappings could be used to visualize the VST structure, instead of the node-link tree layouts we chose for. Examples are classical and Voronoi treemaps. While such representations are more compact than node-link tree layouts, they also have some limitations. First, space-filling hierarchy visualization methods dedicate, by construction, most of the screen space to showing the hierarchy. This does not leave enough space to show the data attributes, like done with our image thumbnails or tag clouds. Secondly, comparing items in terms of similarity is arguably harder using (Voronoi) treemaps. Indeed, in such treemaps, the positioning of the items would respect the original distances even less than our VST method does. Thirdly, treemap methods focus presentation mainly on the leaves or one level at a time, whereas the node-link layout we chose to use can show multiple levels (from the root onwards) in a tree in the same time. Figure 3.15 illustrates this by comparing visualizations of the *Freephoto* dataset computed using (a) classical squarified treemaps; (b) Voronoi treemaps; and (c) our proposed VST. Treemap images were generated using the software available at http://www.treemap.com/. Here, we used only the first two levels of the tree hierarchy – using more levels is technically possible, but more levels are hard to see in the treemap. Compared to the VST image, we argue that the treemap images offer a more abstract, and harder to understand, view on similarity between items. Additionally, it is hard to adapt treemaps so that they take into account the edge weights of the VST that encode similarities. However, on the other hand, treemap views make it easier to find the supernode that a lower-level node is part of. This can be done using the VST too, but entails either displaying two tree levels and correlating them in linked views using interaction, or visually separating a lower-level tree layout into visual groups that correspond to separate branches.

The relative pro's and con's of node-link views and treemap views suggest that the two techniques could be combined. For example, treemap-like views could be an alternative for the exploration of higher levels of a VST, since one is typically not interested for assessing exact similarity at such levels. For lower levels of the exploration, one could switch to the node-link views.
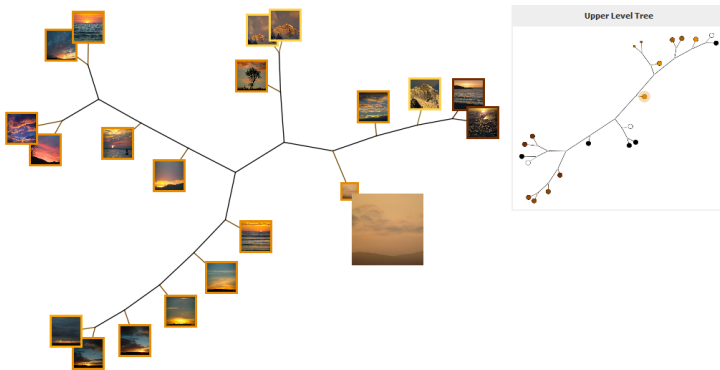
VSTs also have several limitations. First, the clustering technique being used may or may not generate an optimal hierarchical view of the data at hand. The

Class

Min ▬▬▬▬▬▬▬▬ Max
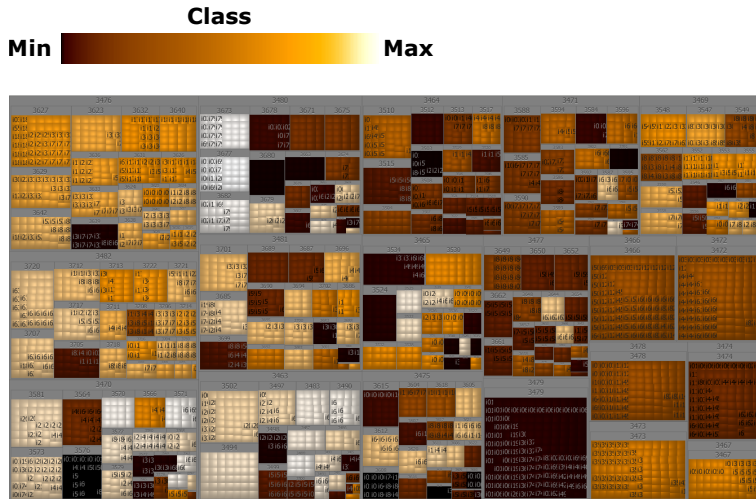
(a) Treemap view of the freephoto dataset.
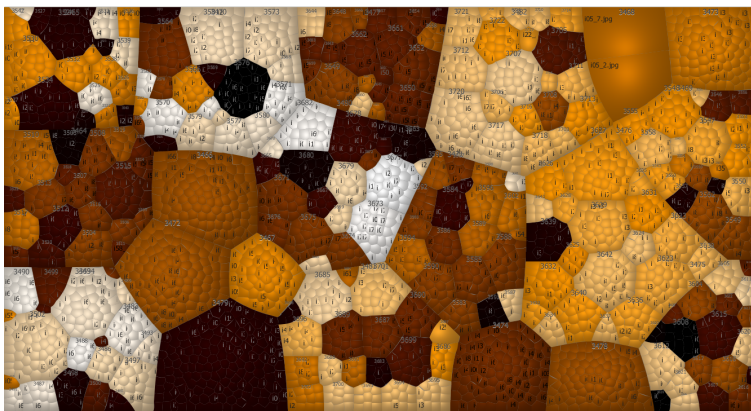
(b) Voronoi Treemap view of the freephoto dataset.

(c) VST top and second level views of the freephoto dataset.

Figure 3.15: Treemap, Voronoi Treemap, and VST views. Although more compact, the Treemap views do not allow the distinction of levels of similarity within a group.

issue is particularly visible for datasets where the distribution of inter-observation distances is relatively flat. Since clustering draws 'hard' borders between groups, and since the resulting tree is visualized with a node-link layout that tries to avoid overlap of different subtrees, the actual *screen space* distance at which points are placed may not reflect well the *data space* distance of the observations. Secondly, clustering results depend, as explained, on clustering parameters, and it is not evident which are good settings for these for a given dataset and problem at hand. In terms of visual encoding, limitations concern the relative simple summarizations we offer so far (one color-coded attribute value, one image, or a small set of tag words), and the lack of direct encoding of the actual data distance values. Edges of the node-link layout could serve as a potential support for adding such information, by using color and/or thickness encoding, along the metaphor presented in [148]. Additionally, nodes could be used to convey the sizes, in terms of numbers of observations, and/or nested tree levels, that the nodes contain. Scalability-wise, our implementation could be further optimized to handle a fast processing of image and text information, and a multithreaded or distributed implementation of NJ clustering could speed up the VST construction.

3.6   Conclusions

We have presented the Visual SuperTree (VST) a strategy for fast similarity-based visualization of large datasets by developing a multiscale similarity tree. Our approach is suitable for any dataset where similarity can be encoded as a distance matrix, thus covers implicitly multidimensional datasets. VSTs reach interactive display and navigation times during the exploration of millions of observations by pre-clustering the data and forming a similarity tree, which is computed on smaller subsets in the various levels of clusters of the original data. The tree building time is dominated by the time to run the clustering algorithm, but the clustering organization and the disk access procedures designed to realize the strategy offers very fast exploration of the results. Quality-wise, VSTs offer a slightly lower neighborhood preservation than classical neighbor-joining trees, but massive accelerations in terms of computation time, thus scalability.

VTSs are amenable to implementation in small-display and large-display devices, yielding full and partial views as required by the device restrictions and capabilities. Our approach extends the capabilities of currently available similarity-based visualizations to the next level of scalability, given as input any multidimensional data set or a similarity relationship between observations of any kind. By partitioning also the visual space, the VST lends itself to the exploring of data that can be summarized, such as image and text datasets.

Several interesting extension directions are possible. First, as already mentioned in Sec. 3.5, several computational optimizations can be done to make the generation of the VST faster and thus more scalable. Secondly, existing similarity data can be encoded more explicitly atop of the VST layout, and more information-rich

summarizations can be thought of. Another very promising direction is to combine multidimensional projections, treemaps, and the VST metaphor in a single hierarchical similarity based visualization of large datasets. This would combine the compactness advantages of treemaps, the more exact distance-preservation capabilities of projections, and the high scalability and intuitive display advantages of VSTs in a single powerful metaphor. Finally, VSTs could be extended to handle richer multimodal datasets, such as combined text, image, audio, and video data.

Similarity Trees for Data-Driven Edge Bundling

4

Graphs or networks are present in a wide range of problems, being useful to model different kinds of relationships between elements [202, 80]. The visual representation of datasets modeled as graphs is frequently used in data analytics. Here, the classical node-link metaphor is (still) the most frequently used, given its intuitiveness and also familiarity. However, graph visualization presents several challenges. One of the most salient challenges relates from the increase of visual clutter when large graphs are drawn, which in turn reduces the power of data analysis. Clutter reduction is a frequent thread in different areas of study in information visualization [53]. The general solution to it is to reorganize, aggregate, and/or transform the visual elements that encode the graph so that the attained representation can reveal patterns that are hidden and/or cluttered in a raw (node-link) representation.

At a high level, the problem of visualizing large graphs with reduced clutter is directly related to our goal of visualizing large similarity trees, which was discussed in Chapter 3. In there, we showed how a mix of data aggregation, suitable visual encodings, and interactive navigation can effectively and efficiently display large similarity trees on a multiscale. In this chapter, we essentially take the same requirements (efficient, effective, and multiscale display of relational data) but extend them to the context of a general graph. What differs, thus, is the input data (a general graph *vs* a tree) and the technical solutions used to address the problem (to be discussed next). What is similar, is the fact that we extract such graphs from datasets that can be represented in terms of a distance matrix, just as we did for the visual supertrees discussed in Chapter 3.

Among the graph visualization techniques, *edge bundling* has obtained great success in the simplified presentation of large graphs in terms of a modified node-link metaphor [213]. The main goal of this technique is to transform a straight node-link graph representation by bending and aggregating edges. Edge grouping reduces edge crossings and increases the amount of whitespace, thereby arguably reducing visual clutter and making the visual analysis of a graph easier (Sec. 2.3.2). Additionally, the smoothness provided by the curved bundled edges also helps improving the data analysis in the visual representation by helping to follow high-level edge patterns, in line with the Gestalt continuity principle [175]. Among the most famous such techniques, Holten [82] introduced Hierarchical Edge Bundling (HEB) to visualize relationships between software artifacts. This technique uses the inherent containment hierarchy of software data to guide the bundling, joining edges that follow the same hierarchy. However, there is no discussion about building hierarchies from *unorganized* data.

Further on, many other edge-bundling techniques have proposed different ways to group (bundle) edges in general graphs, where no hierarchy is explicitly present. Examples include strategies based on force-directed edge bundling (FDEB) [84, 165], geometry processing [35, 108], clustering [64, 16], and image processing [185, 56, 86]. A key commonality of all these techniques is the usage of *spatial* information to do the bundling. While this effectively simplifies a *drawing* that reflects the data, it does not produce a simplified drawing of the *data*. Recently, the use of data to perform bundling has gained some attention [147, 73, 209, 178]. However, such techniques are, we argue, mainly adaption of earlier *spatial* bundling methods to incorporate data elements, and not techniques for *directly* visualizing data that has no given spatial embedding.

Given our research interest in generating simplified views of large datasets which reflect the *similarity* of data elements introduced in Chapter 1, edge bundling is an interesting candidate for analysis and extension. In this chapter, we extend and adapt HEB in order to construct bundled graph layouts for visualizing similarities of large collections of unstructured data in a simplified way. In contrast to existing bundling techniques, we consider data points that do not have a given spatial embedding, and are characterized only in terms of a similarity function. We first employ a multilevel clustering approach to aggregate the most similar nodes, similar to the construction of the VST discussed in Chapter 3. Next, we compute a high precision similarity tree that guides the bundling process and faithfully represents the high-level structure of the multilevel data representation. We visualize the resulting bundled graphs using a radial layout representation, as in HEB, and also using a force-directed graph layout, as in FDEB. We also propose a multiscale visualization which enables the exploration on multiple levels of detail of a given dataset. We evaluate and compare our technique *vs* state-of-art bundling techniques in terms of how meaningfully the bundling can represent high level data patterns in artificial and real datasets.

Summarizing the above, our main contributions here are:

- A *framework* to build bundled-edge layouts from similarity measures defined on an unstructured dataset. This framework, we argue, can meaningfully simplify the similarity-based visualization of such datasets;

- A *summarization* of bundled-edge layouts that takes into account data-based similarities. This summarization improves the visual and computational scalability for exploring large datasets;

- Several *applications* of our technique to visualize different kinds of data, such as paper citations and dynamic tweets.

We structure this chapter as follows. In Section 4.1 we give an overview of edge-bundling state-of-the-art techniques that refines the discussion presented in Sec. 2.3.2. In Section 4.2 we describe our bundling technique as well the simplification procedure we propose to enable a simplified multiscale navigation through

the data hierarchy. Section 4.3 presents analysis and comparisons with state-of-art bundling techniques on several datasets. In Section 4.5 we discuss our technique. Section 4.6 concludes the chapter.

4.1 Related Work

A brief introduction of edge bundling is given in Section 2.3.2. Below we extend this review from the perspective of bundling seen as a method to visualize relational data of an increasing amount of *diversity*.

TREES: The original idea, proposed by Holten [82], and called Hierarchical Edge Bundling (HEB), is basically a method to produce a simplified visualization of relational datasets that are organized as trees. Such data is sometimes also called a *compound* graph [44, 45], as it consists of two types of relations: hierarchical ones (that define the tree) and other relations (that define the relational dataset proper). HEB employs the tree information to settle paths to guide the bending and grouping of the relations. This is performed drawing each edge as a B-Spline curve taking as control points the intermediate points of a given hierarchy. The proposed strategy is fast since the vertices and control points are fixed during the drawing phase, so reducing the entire process to drawn B-Splines curves. However, it relies on an inherent hierarchy as input data, and consequently cannot be applied to scenarios where the input data is of a different type.

GRAPHS: Recognizing the limitations of HEB, following approaches have considered relational data that comes as general graphs, *i.e.*, contains nodes that are not organized in any hierarchical fashion. Different approaches have been proposed to tackle this problem, avoiding the need of any other information besides the graph adjacency and the node positions to perform the bundling. A first example here is Force Directed Edge Bundling (FDEB) [84]. FDEB essentially creates a system of forces that attract close points along close edges in a straight-line graph drawing. Thereby close edges are bent until they group to form bundles. The Divided Edge Bundling (DEB) [165] technique improves the FDEB layout by separating edges with different directions, improving readability for directed graphs. The major problem of force-based strategies is their high computational cost, due to the need of finding close edge fragments in a large changing set of curves.

Another group of techniques are geometric-based approaches. One example is Geometry-Based Edge Blustering (GBEB) [35]. GBEB builds a mesh based on the straight-line graph drawing and uses this mesh to reason about the relative position of close edges to guide bundling. Although a suitable solution, the mesh construction is a complex process with high computational cost. Winding Roads (WR) [108] addresses this limitation using a hybrid approach that combines quadtree decompositions and Voronoi diagrams to accelerate the various proximity queries that are needed to find and route close edges.

Recognizing the computational cost of general-graph bundling as being a problem, image-based techniques have next been developed. These reduce this cost by suitably modeling proximity queries and the displacement of close edges to a common central location in terms of image processing operators, which are highly parallelizable. A first technique in this group is Image Based Edge Bundling (IBEB) [185]. IBEB does not actually bundle graphs, but proposes a way to create multiscale simplified drawings of already bundled graphs. The Skeleton-Based Edge Bundling (SBEB) [56] technique finds bundle locations as the centers of groups of close edges, which are computed by medial axis techniques. Kernel Density Estimation Edge Bundling (KDEEB) [86] finds the same locations by an iterative sharpening of the density of edges in a graph drawing, essentially recasting the well-known mean-shift technique used in image segmentation [30], which is much faster than previous methods. A further speed-up is proposed by CUBu, which implements the KDEEB process fully on the GPU in the CUDA programming language, achieving the so-far fastest general-graph edge bundling technique known [194] technique. Besides image-based techniques, speed-ups are also achieved by using a hierarchical approach, similar to well-known techniques for graph layout computations [77]. For example, Multilevel Agglomerative Edge Bundling (MINGLE) [64] aggregate the edges based on an ink-minimization concept. The algorithm iteratively groups the closest edges until the amount of ink used to create the visual representation reaches a minimum. This process can be viewed as a edge-simplification strategy, where groups of edges are aggregated and represented as single edges. After simplification, curved lines connect these aggregated edges to the original vertices. Although this simplifies the graph drawing and is relatively fast to compute, such layouts are less readable due to the lack of a clear definition of the main edge patterns [86].

ATTRIBUTED GRAPHS: All previous techniques can be seen as purely geometric, in the sense that they aim to produce a simplified view of a graph *drawing*, and not of a *graph* itself. However, in practice, graphs often encode additional data, for example in terms of node and/or edge attributes. A few recent techniques recognized this limitation and proposed to incorporate attribute data. The Attribute-Driven Edge Bundling (ADEB) [147] technique extends KDEEB by using edge attributes define which edges are allowed to be bundled. However, besides direction, only a single other attribute is used. Yamashita and Saga [209] extend the FDEB technique by using the edge type or attributes on the force calculation. In both cases, only edge information is considered, and information contained on the vertices are still ignored.

OTHER DATASETS: Besides graphs and trees, bundling has been also applied to other datasets. The most salient example are trails, or trajectories, of vehicles and of eye tracking [88, 147]. In both cases, however, the dataset to be bundled is still spatial.

BEYOND SPATIAL DATA: Given the proven ability of bundling to produce simplified views of relational data, It is tempting to consider its application to non-spatial data. More precisely, since bundling's core ability is to simplify a drawing based on spatial similarities in the drawing, we are interested to see how we can simplify a dataset, based on the similarity relations between its elements, by constructing a suitable spatial embedding of these similarities, and next adapting bundling to simplify this embedding. This would allow bundling to handle a much more diverse set of data types.

A separate direction of study is how to extend the simplification power of bundling. The bundling operation itself can be seen as a multiscale operator that covers the entire range from a fine-scale representation of the data (unbundled view) to a coarse-scale representation (fully bundled view). This is also explicitly seen in the iterative structure of many existing bundling techniques [84, 56, 86, 64, 194]. However, this multiscale is purely *spatial* – the number of data elements being shown stays the same at all simplification level, and the simplification is simply achieved by overdraw. We consider to add to this simplification in the *data* space, by providing a multiscale representation of the actual data. When combined to bundling, this will yield a multi-resolution representation of bundled data, that is arguably able to cope with much larger datasets in terms of reduced visual clutter. This will enable the exploration of a similarity-based relational dataset on different levels of detail, from a more abstract to a more concrete view, where coarser levels are better suited to explore the main patterns between groups, and finer levels are better suited to verify intra and inter groups relationships.

4.2 Similarity-driven Edge Bundling

A multidimensional dataset of N observations can be modeled as a matrix $X = \{x_1, \ldots, x_N\}$, in which each observation $x_i \in \mathbb{R}^n$ is defined by a n-dimensional vector of quantitative attributes. For data such as images or text, represented by extracted features, the dimensionality n of observations can range from tens to thousands.

As outlined earlier, we aim to visualize the similarity structure of such a dataset. To capture this similarity, we can use a distance matrix $D = (D_{ij})$ where $D_{ij} \in \mathbb{R}^+$ encodes the distance between observations x_i and x_j. Such distances can be computed from the attribute values of the two observations, as outlined in Sec. 2.1.2.

Distance matrices can be further represented in a simplified form by similarity trees, such as Neighbor Joining (NJ) trees [157]. NJ trees have been successfully used in the context of multidimensional data exploration [34, 136, 57]. NJ trees group similar observations under the same parent recursively. We describe the construction of such trees from multidimensional data in Sec. 4.2.1. Further on, such trees can be further simplified, and computed faster, by pre-clustering the observations, as described in Sec. 4.2.2. We use such multilevel NJ tree represen-

tations to guide the bundling of similarity relations to obtain our final simplified visualization of the data similarities, where bundles connect highly similar observations and observation groups, as described in Sec. 4.2.3. Figure 4.1 shows our entire pipeline, whose steps are detailed next.
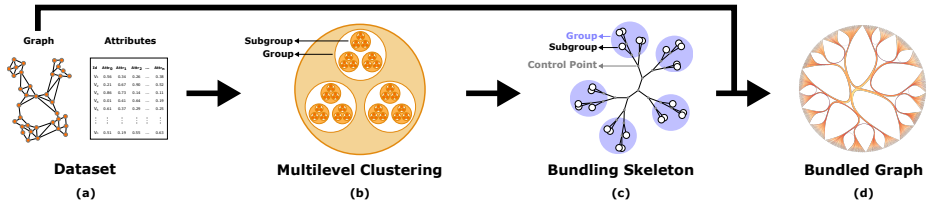

Figure 4.1: Similarity-based data visualization using bundling.

4.2.1 Similarity Tree Construction

A phylogenetic tree describes evolutionary relationships between entities. They are positioned in a hierarchical model where entities that belong to the same evolutionary descendancy are kept on close neighborhoods. The process of constructing the true topology to represent the minimum evolutionary distance is known to be NP-hard, since all possible topologies have to be explored. However there are algorithms that build approximations in polynomial time, in which the Neighbor-Joining (NJ) [157] is one of the most widely used. The NJ algorithm has been described in Sec. 3.2.1. As explained there, the input for the NJ algorithm is the distance matrix D (defined also at the beginning of Sec. 4.2), and the output is a tree having the N observations in the input dataset as leaves and other $N - 2$ non-leaf nodes.

The visual encoding of a phylogenetic tree is intuitive and easy to explore. Groups of similar observations are easily spotted by examining small neighborhoods of tree branches, allowing the visualization a certain freedom in placing nodes on the visual space, unlike when using multidimensional projections, where the screen-space distance between points is the only encoding of their data-space distance (Sec. 2.4). This way, phylogenetic tres can be visualized using several graph layout approaches [19, 7], which might each highlight different data patterns. The NJ algorithm offers good precision, and it has been applied in information visualization with great success [34, 136]. We also describe several applications of visualizations based on the NJ tree idea in Chapter 3.

Since non-leaf nodes join similar nodes, branches formed by such nodes can be used to define a 'skeleton' to the bundling process, much like the given hierarchy data is used to define a skeleton for the HEB bundling [82]. Here, non-leaf nodes will serve as control points for routing graph edges, which in turn encode inter-observation similarity relations. This process is further described in Sec. 4.2.3.

4.2.2 Multi-level Aggregation

Due to the cubic computational complexity of the NJ algorithm, it is not well suited to process large datasets. Handling datasets larger than a couple of thousand observations takes a long time demands large amounts of memory to fit the distance matrix. Additionally, there is a high number of non-leaf nodes present in a typical NJ tree that are part of the same leaf-to-leaf branch – if the tree is balanced, this is $O(\log_2 N)$ for N leafs, but can be as large as $O(N)$ for unbalanced trees. This creates several potentially very long paths with many control points, which in turn can yield on distracting wiggles in the resulting bundles. As such, large NJ trees are not always suitable as skeletons for bundled visualizations. Finally, our aim of providing a multiscale (coarse to fine) visualization requires the ability of showing NJ trees at various levels of detail.
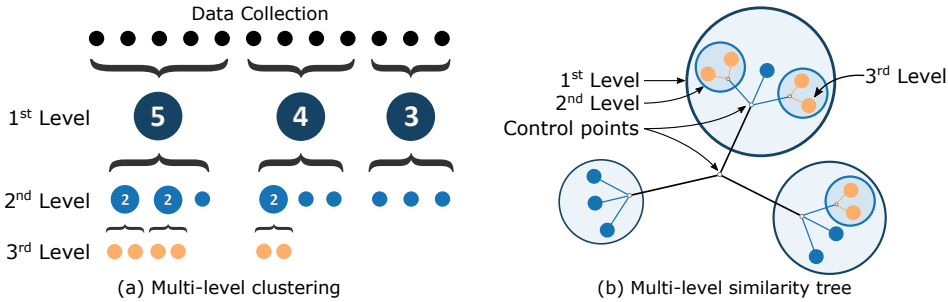
Figure 4.2: a) Multi-level clustering using $k = 3$ clusters per level and $s_{max} = 3$. b) Multi-level similarity tree constructed from the clustering.

We jointly attack these problems by employing as similarity tree the Visual SuperTree (VST) described in Chapter 3. To construct it, we first cluster the input dataset X to create k clusters $C^{0,0} = \left\{ c_1^{0,0}, \ldots, c_k^{0,0} \right\}$. Next, we recursively split each cluster $c_i^{0,0}$ larger than a user-given value s_{max} using the same clustering procedure as for the first step, yielding a partition $C^{j,1} = \left\{ c_1^{j,1}, \ldots, c_k^{j,1} \right\}$, and add all $c_i^{j,1}$, $1 \leqslant i \leqslant k$ as children of $c_i^{0,0}$, for a given i. Together, these elements form a cluster tree (not to be confused with the NJ tree). Here, $c_i^{j,1}$ defines the ith element of cluster j on level l in this tree. The procedure stops when only clusters smaller than s_{max} exist. Figure 4.2 illustrates an example of multilevel clustering using $k = 3$ and $s_{max} = 3$ for a dataset of 12 elements.

Next we build a tree T^l for each level l of the clustering hierarchy. The nodes of T^l are clusters in $\{C^{j,l}\}$, $\forall j$. Concretely, on the lowest level, we create a phylogenetic, or classical similar, tree connecting the individual observations in a cluster. On the upper levels, we create a similarity tree connecting clusters, using clusters centroids as representatives to create the distance matrix needed for the NJ algorithm (for details, see Sec. 3.2.1). Hence, each level l of the clustering hier-

archy, from the top to the bottom-most one, generates an increasingly larger tree $|T^l| < |T^{l+1}|$, $\forall l$, representing similarity on an increasingly finer scale.

The set $\{T^l\}|$ of similarity trees is next used to produce a simplified view of the data similarities at various level of details. This is described in the next section.

4.2.3 Graph Drawing and Bundling

To display similarities using the set $\{T^l\}|$ of similarity trees and bundling, we use a given tree T^l, or combination of parts of several such trees, to bundle edges between the original observations. This generates a different notion of *scale* from the approach of using NJ trees presented in Chapter 3: There, when selecting a coarser scale, the number of displayed data elements is decreased, as groups of related observations are replaced by their enclosing supernode. In the method described in this chapter, selecting a coarser scale uses a coarser similarity-tree T^l to display similarities, but these still relate full set of lowest-level observations. In other words, the method in Chapter 3 proposes a notion of scale for both data items and their similarities, whereas the method presented in this chapter proposes a notion of scale only for similarities. Using the full set of observations is motivated by the fact that bundling achieves the desired visual simplification without any additional reduction of the number of displayed nodes. This was not possible when drawing relations using straight node-link metaphors as we did in Chapter 3.

The selection of scale can be done interactively, as follows. We usually start by using the coarsest similarity tree T^0 implied by the coarsest-level clustering $C^{0,0}$. We next arrange all leaf nodes (observations) following this tree (as described further below), and then bundle observation relations between all these nodes along the branches of T^0. This produces a coarse-level, simplified, view of the edges in the dataset, guided by a structure that reflects similarity of observations. Next, we can refine the exploration of similarities, either by replacing T^0 by T^1 (uniform refinement of the level-of-detail), or by selecting a specific cluster $c_i^{0,0} \in C^{0,0}$ to refine (local refinement of the level-of-detail). In the latter case, we retrieve the similarity tree that is contained within $c_i^{0,0}$ and replace the corresponding node in T^l by this tree. This yields a 'mixed' similarity tree which describes different parts of the dataset at different levels of detail. We then use this mixed tree to bundle edges between observations, as described earlier. The process of navigating the scale of similarities is done interactively, as described next in Sec. 4.4.1

We describe next the technical elements that construct the bundling process itself, once a (mixed) similarity tree has been chosen by the procedure described above. For this, we need two operations: placement of the nodes to be connected by bundled edges, and the bundling of the edges themselves.

Node placement: For this step, we take advantage of the NJ tree structure, and use a tree drawing algorithm to lay out the chosen similarity tree. As we want

to show edge bundles linking all lowest-level observations, we replace, in this tree, any leaf cluster node by its individual observations. This way, the leafs of the resulting tree will always be raw observations. As a tree layout algorithm, we experimented here with radial and force directed algorithms. These have been described earlier in Sec. 3.2.2.

Edge bundling: We consider here all edges connecting leaves of the above-mentioned tree. For each such edge, we construct its control polygon by searching for the shortest path Π in the similarity tree between its endpoints. This path will include non-leaf nodes in the similarity tree. Given such a path, we route the drawing of an edge along the path's control points using a B-spline curve, as in the original HEB algorithm [82]. Next, we construct discrete finely-sampled polyline representations of these B-spline curves, and render them with suitably chosen colors and transparencies, again following the guidelines of the original HEB algorithm.

We also allow the user to control the tightness of the bundling, again, following the HEB technique. For this, we introduce a parameter $\beta \in [0, 1]$ that controls the movement of the points of the polyline representing the fully-bundled drawing of an edge towards the corresponding points of the sampling of a straight line connecting its endpoints. Denoting the fully-bundled drawing of an edge by the polyline $\{\mathbf{b}_i\}$ and the corresponding straight-line connecting the edge's endpoints by $\{\mathbf{s}_i\}$, $1 \leqslant i \leqslant S$, where S is the number of sample points, the positions of the relaxed points $\mathbf{q}_i$ that we ultimately use to render the edge are given by

$$\mathbf{q}_i = \beta \mathbf{b}_i + (1 - \beta)\mathbf{s}_i. \tag{4.1}$$

Values of β close to 1 produce less bent edges, thus an image closer to a straight-line rendering, which has however more clutter in terms of edge intersections. Values of β close to 0 produce more distorted (bent) edges, which however reduce clutter and enhance visual separation by increasing the amount of whitespace between bundles. As such, we allow users to control β interactively, like in many other bundling applications [82, 84, 86, 56, 194].

However, changing the bundling strength β can highly deform short edges which have many control points in their paths. In order to avoid this behavior, we propose to use an adaptive β setting, where each edge has its own β value, instead of use the same global β for all edges. The per-edge β value is based on the length and weight of each edge. Concretely, we apply less bending (use higher β values) for short and low-weight edges to make them straighter and thus easier to follow end-to-end in the visualization. We note that a related special treatment of short edges was also proposed by CUBu [194]. There, the authors observed that short edges can often be obscured in large bundled graph drawings – since they are short, they have a very low chance of standing out in bundled visualizations, as they easily get 'sucked in' larger bundles. Long edges have this problem less, since they themselves determine the appearance of long, salient, bundles. The solution proposed there was to modulate the opacity and/or color of edges based

on their length, to emphasize short edges. Our solution addresses the same goal
– increasing the visibility of short edges – but the proposed solution is different:
Rather than changing the opacity or color of edges, we simply change their bun-
dling strength. This way, short edges bundle less, thus get a higher chance to stand
out in the final visualization.

4.3   Comparison

In this section, we compare our method, which we dub Similarity-Driven Edge
Bundling (SDEB), with other well-known edge bundling algorithms in the liter-
ature. The main goal of this comparison is to determine the differences between
SDEB and other algorithms in terms of produced results, and next to analyze
these differences and reason about the added-value of our method versus existing
methods.

Evaluating edge bundling algorithms is a complex task. The key problem here is
that one does not have, in general, a 'ground truth' image that would be the ideal
one produced by a bundling technique. Hence, when evaluating a concrete tech-
nique, it is hard to say (a) which patterns in the displayed image are correct (true
positives), (b) which patterns in the displayed image are misleading since they
do not exist in the data (false positives), and (c) which patterns in the data are
not reflected by the displayed image (false negatives). Moreover, such evaluations
depend strongly on the types of patterns that one is interested to find.
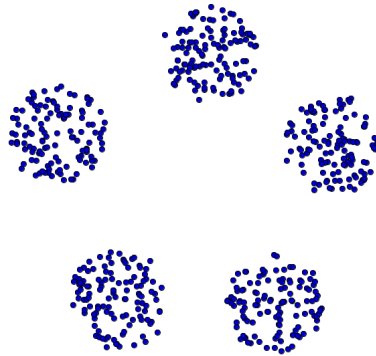
Figure 4.3: Synthetic dataset of 5 clusters projected by Multidimensional Scaling.

This problem is well-recognized in many publications on edge bundling. As
such, most such publications propose to evaluate bundling by comparison and
with respect to a set of generic aesthetic criteria. In detail, bundling is evaluated
in terms of visually comparing bundling images constructed by different algo-
rithms on the same dataset; and the comparison involves checking for desired vi-
sual properties such as smoothness of bundles, clear separation between bundles,
ease of following a bundle end-to-end, and reduced deformation of the original

paths [213, 185, 56, 194]. It is argued that if a bundling method *A* respects such properties better than a bundling method B, the *A* will arguably better show the underlying data patterns than B for the same dataset.

We take the same approach here for evaluating SDEB. For this, we select the following methods to compare it against: FDEB [84], MINGLE [64], and CUBu [194]. This selection is motivated by several factors: These methods are designed to work on general graphs; they are well known in the literature; they can handle large graphs; and they are easy to implement or we have access to a standard implementation.

The second element in our comparison is to select a dataset. We chose here to create our own synthetic dataset, so as to be able to control the type of patterns that it contains. This way, we can judge much better whether a bundling method is more faithful than another method than if we used a real-world dataset for which we do not have a clear understanding of the involved data patterns. In detail, we created a synthetic dataset containing 600 multidimensional elements that are separated into five groups. Figure 4.3 shows a projection of this dataset using Multidimensional Scaling (MDS).

These elements will constitute our nodes in the bundled image. As similarities, we consider the high-dimensional inter-point distances. From this data we generated 4000 edges, and then we simulated three edges distributions for our comparison experiments, as follows:

1. several connections between vertices from two compact groups, and few connections between others vertices;

2. several connections between vertices from one compact group directed to vertices on two compact groups, and few connections between others vertices;

3. several connections between vertices of the same group, in two different groups, and few connections between others vertices;

Figure 4.4 shows the comparison of the layouts obtained with SDEB, FDEB, MINGLE, and CUBu. Each row of the figure shows images computed with the four considered layouts. Consecutive rows show the usage of a radial *vs* a force-directed node-placement layout. Edges are colored based on their lengths: dark blue for short edges, up to light green for long ones, with transparency adjusted to help on the identification of regions with dense concentration of edges.

Before analyzing the results, one important observation has to be made. The effectiveness of our type of visualization in showing groups of similar nodes is affected by two components: the placement of *nodes* and the *bundling* being used. In our case, nodes are placed by using a drawing of the similarity tree, either by a radial layout, or by a force-directed layout, as explained earlier in Sec. 4.2.3. It can, of course, be argued that other node placement techniques are better for our goal of highlighting similarities. However, we cannot analyze an open set of
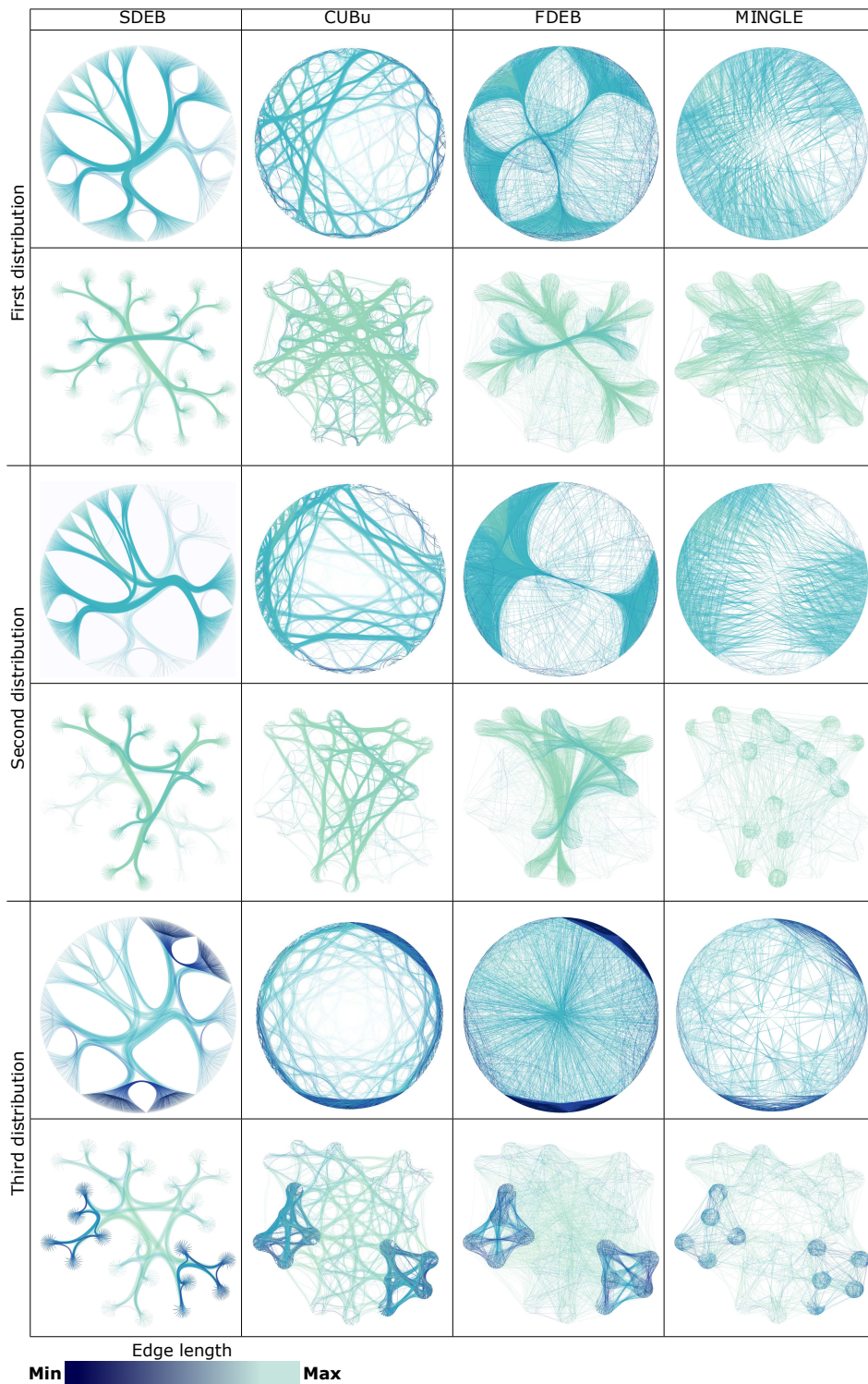
Figure 4.4: Comparison of bundling techniques on three edge distributions, using radial and force-directed node placements.

such algorithms, so we have to make choices. Secondly, by fixing the node place-
ment and varying the type of bundling, we can compare the bundling algorithms
themselves. It can also be argued that a certain bundling method would give bet-
ter results when used in combination with a different node placement algorithm.
Such studies, while very interesting and valuable, are however out of our scope,
and also of the scope of most evaluations of bundling methods known in the lit-
erature. As such, we fix the node placement and vary bundling techniques in our
evaluations, and leave the more complex joint evaluation of bundling-and-node-
placement for future work.

Several observations can be made in Figure 4.4. First, we see that the considered
methods yield dramatically different pictures for the same input dataset. We also
see a large variation in the amount of structure (or alternatively clutter) created
by different methods. Overall, FDEB and SDEB show the coarse-scale similarity
patterns which stands out from the remainder of the graph. CUBu is able to show
finer-scale similarity patterns quite well, but cannot highlight the coarsest-scale
ones, yielding an 'organic network'-like effect where bundle crossings create am-
biguities when following bundles end to end. MINGLE has, in general, the least
success in creating well-delimited bundles and in simplifying the visualization, an
effect also seen in other evaluations [194]. In terms of edge smoothness, the meth-
ods can be ordered as: MINGLE (smoothest, but too smooth, as bundles do not
form well); FDEB; SDEB; and CUBu (very well formed fine-grained bundles, but
too many ondulations). Methods also vary with respect to their interaction with
the node placement being used: SDEB and CUBu arguably show similar quali-
ties of bundles for both the radial and force-directed placement, while FDEB and
MINGLE produce low-quality results when the radial layout is used. Overall, we
believe that SDEB strikes a good balance between the simplification of the image,
ease of following bundles, and independence on the underlying node placement
algorithm.

4.4 Enhancements

Our proposed simplified similarity visualization can be enhanced in several di-
rections. We present next two such directions: multiscale interactive exploration
(Sec. 4.4.1) and exploration of dynamic datasets (Sec. 4.4.2). This discussion also
introduces some real-world datasets and questions that we target with our visual-
ization.

4.4.1 Multi-level exploration and summarization

A graph representation is frequently used to model collaboration among differ-
ent researchers. A simple model for organizing documents (articles) is here the
so-called *authors network*, where nodes indicate articles (documents) and edges
indicate articles co-authored by the same persons. Other models for organization

of articles are known, such as using relations to model citations between papers. For this application, we consider the Visualization publications dataset, which contains publications in the IEEE Visualization conference in the period ranging from 1990 to 2015. Entries in this dataset consist of papers, attributed by their author names, title, abstract, citations (within the same dataset), and year of publication, among others. A full description of the dataset is available online [91].
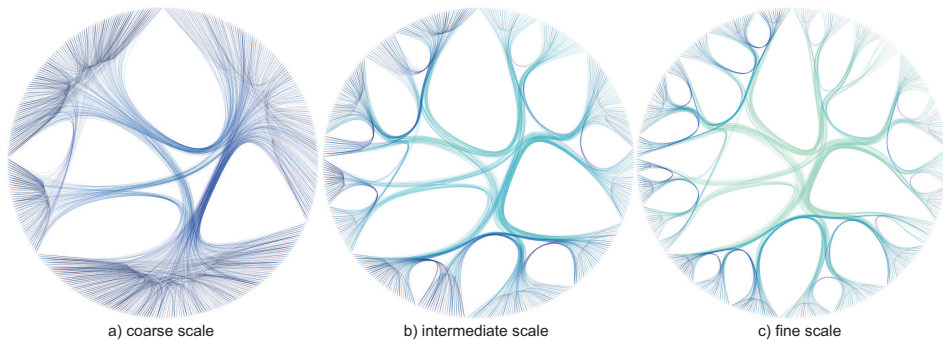

| a) coarse scale | b) intermediate scale | c) fine scale |

Figure 4.5: Exploration of similarities of the IEEE Visualization papers dataset on three different scales.

In our application, and in contrast to collaboration networks, we focus on visualizing the similarity structure of the papers. For this, and in order to build the similarity tree, we first convert all papers into multidimensional vectors following the bag-of-words representation [158]. Next we conduced an experiment to cluster this multidimensional datasets, checking which parameters would present the best data distribution for the k-means clustering technique. We obtained $k = 4$ and the maximum cluster size being $s_{max} = 100$. Then, we applied the SDEB algorithm as described in Sec. 4.2.

Figure 4.5 shows three different scales of the resulting visualization, ranging from coarse to fine. Edges are blended and colored using a color map that depicts similarity – dark blue values show connections between the most similar nodes (shorter) and light green values show connections between the least similar ones (longer). The number of leaf nodes (papers) is the same in all three cases. The first scale (image (a)) shows an overview of this dataset. We see here a clear separation of the four aforementioned clusters, and also the balance between intra-cluster and inter-cluster relations. We see here that papers are organized, similarity-wise, into one smaller group and three large groups, with the smaller group densely connected with two other group. As we explore finer scales (images (b,c)), we can see a finer-grained grouping and the resulting edge patterns.

Taking advantage of the hierarchical structure, we propose an interactive summarization mechanism. Here, one can select groups of similar vertices, *e.g.* by selecting closely-placed nodes in the visualization or by selecting an entire subtree, including non-leaf nodes. Next, these nodes are collapsed into a single supernode.

When doing this, we hide all vertices of the lower hierarchy of the selected group, and we display only the root vertex of the selected hierarchy. Next, in the space left blank by this hiding in the layout, we display topics for the textual data associated to the selected leafs (papers), using a tag cloud technique. Figure 4.6b shows this. The left image shows the fine-scale visualization of the papers dataset, and is similar to Fig. 4.5b. The right image shows a summarized visualization, where the user has selected three separate groups of papers to be summarized. For clarity, each group is rendered in a different color. Keywords (tags) are retrieved based on the relative frequency of the words in the papers in a group, by selecting a few most-relevant such words [143]. The sizes of the tag bubbles maps the frequency (importance) value.

(a) Non-summarized dataset   (b) Summarized dataset
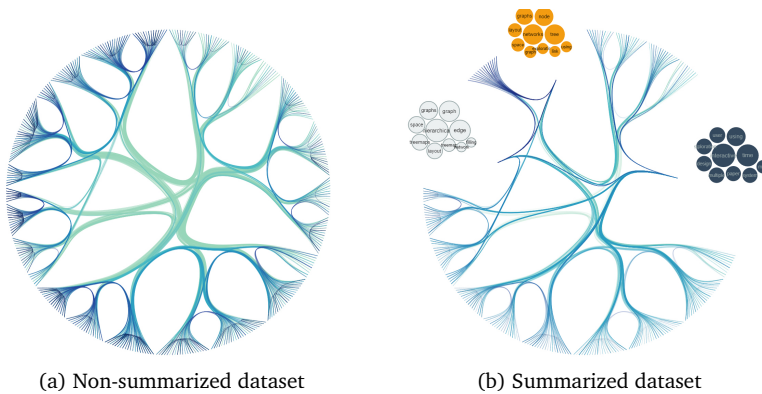
Figure 4.6: Summarization of selected groups in a SDEB visualization of the IEEE Visualizations papers dataset.

The similarity tree that serves as skeleton for bundling in SDEB allows additional layout variations to be designed. One of these addresses the problem of many bundling algorithms, including the standard SDEB, that visually tracing bundles end-to-end can be hard when many hierarchy levels exist. Indeed, in such cases, there are many bifurcation points of the bundles, and since these have various turns, it can be hard to visually follow the path of a specific bundle.

To address this, we propose to remove a part of the 'central' control points along an edge. These are control points which correspond to higher-level nodes in the similarity tree. We proceed as follows. For a given level of detail of the SDEB visualization, we define for each control point of an edge the distance (in terms of similarity tree levels) of the point to the leaf nodes. When this distance exceeds a user-set threshold, the control point is removed, *i.e.*, it will not influence the bundling of that edge. Figure 4.7 shows this method for the finest level of detail of the IEEE Visualizations papers dataset. The images (a-d) show the effect of removing control points at a distance of 1, 2, 3, and respectively 4 from the root of the similarity tree. As visible, the effect is to increasingly 'pull out' bundles to follow
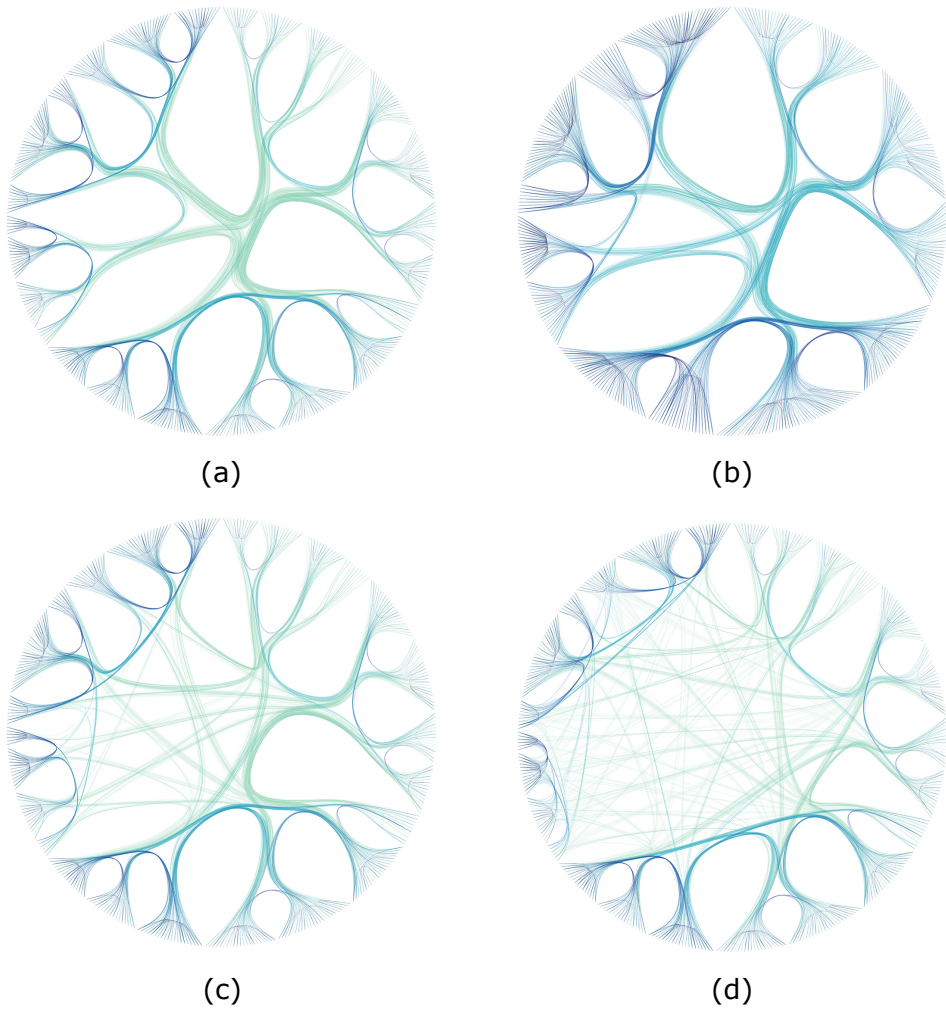
(a)

(b)

(c)

(d)

Figure 4.7: Removal of central control points: (a) Root node; (b) Second level nodes; (c) Third level nodes; (d) Fourth level nodes.

straighter paths between their endpoint groups. This removes part of the bundling structure present in the center of the image, and pushes it towards the periphery. Edges that connect dissimilar nodes (encoded by the light green color) are now weakly bundled, so they become much easier to follow. Edges that connect highly similar nodes (encoded by the darker blue colors) are still strongly bundled. The effect is conceptually similar to a relaxation method where one would use a relaxation factor β which is a function of the similarity (Sec. 4.2.3). However, if we were to do the above, we would obtain a continuously weaker set of bundles over the entire image, as the parameter β varies. This would yield several 'wide' bundles, thus many more crossings which would create clutter. Our proposal does not explicitly weaken the coherence (tightness) of the bundles, but only changes their paths.

4.4.2 Dynamic Graphs

A second extension of SDEB is towards the visualization of dynamic graphs. These are graphs where edges, or both nodes and edges, change in time, in terms of their presence and/or attributes at a given time moment. Dynamic graphs can be further classified into streaming graphs, where each node and edge has a 'lifetime' modeled as a compact interval along the time axis; and sequence graphs, which are discrete ordered sets of graphs with explicit correspondences between nodes and edges from adjacent elements in the sequence, or frames [88]. Dynamic graphs are visualized both by edge bundling methods [88, 129] and other more general graph drawing methods [4, 9].

The advantage of SDEB of letting one control the bundling by a separate data structure – the similarity tree – allows the easy creation of bundled visualizations of dynamic graphs where only edges change. In order to demonstrate this application, we collected a set of publications from Twitter[1], between December 19, 2014 to January 21, 2015, related with the *NBA All-Star Game* polling. This dataset has more than 1.37 million tweets with the tag *#NBABallot*. Each tweet contains, apart from the tag, a timestamp of its publication, and a player names which generates a vote for the respective player in the NBA election. Based on this data, we build a graph connecting players voted by the same Twitter user, yielding 11793 different edges and 426 nodes.

Using the temporal information about the publication of a tweet, we can segment the votes through different frames, each covering a consecutive day during the lifetime of the entire dataset. For each frame, we thus also extract a separate graph built as discussed above, considering only tweets emitted in that time-span. Next, using as similarity function 14 statistics for each player in that season, we constructed a single similarity tree for the entire dataset (all time-spans). Finally, we bundle edges describing players voted by the same Twitter user along this tree,

---

1 https://twitter.com

using sets of edges for the different time-spans based on their publication time of the respective tweets.

For this application we employed the force-directed layout for the similarity tree. The force-directed method has a key advantage in comparison to the radial layout – the latter suffers from the issue of creating too long edges between nodes which are placed far away from each other on the circle's circumference. A force-directed layout ensures that the positions of nodes can better reflect their distances in the tree hierarchy, so that the node-to-node distances in the drawing reflect better the similarities of nodes. Figure 4.8 illustrates our dynamic SDEB bundling. Here, we selected 4 time-frames from the entire dataset, for 4 consecutive days of voting. We notice here how the layout changes smoothly from frame to frame, due to the use of the same 'skeleton', given by the similarity tree. We also notice here several long edges, which connect dissimilar nodes in all considered frames.

4.5   Discussion

The proposed SDEB bundling technique can be thought as being at the cross-roads of HEB and all other general-graph bundling techniques. Similar to HEB, it uses a hierarchy (tree) to route edges into bundles. However, similar to all general-graph techniques, it does not require one to provide such a tree. The tree itself is built from data that comes with a set of observations, by a similarity function. The above make SDEB more general than both HEB and general-graph bundling techniques.

Similarly to HEB, SDEB uses the constructed similarity tree to spatially embed nodes. By using suitable tree layout algorithms, we then get similar nodes (which next will be linked by bundled edges) being placed relatively close to each other in the embedding space. This implicitly reduces clutter and crossings int the bundled image. This is also a property of general-graph bundling methods which embed nodes based on the graph structure – nodes being close to each other in the graph will be placed, in most cases, close to each other in the embedding space, thereby improving the bundle readability. However, many general-graph bundling algorithms are used on graphs where nodes have fixed locations, such as geographical placement [84, 86]. In such cases, bundles cannot be 'shortened', and a large number of crossings will arguably take place.

Formally speaking, SDEB cannot be applied to graphs with given node placements such as the above-mentioned ones, since SDEB assumes to be able to construct the node placement by the execution of the layout of the similarity tree. This restriction can however be relaxed, by using a tree layout algorithm where leaf nodes are fixed and only non-leaf nodes are allowed to move. Creating such an algorithm from, for instance, a force-directed method is quite simple. However, the quality and readability of the resulting bundled images may be low. Indeed, in such images the bundle structure will try to encode similarity of the nodes in

a) frame 1

b) frame 2

c) frame 3

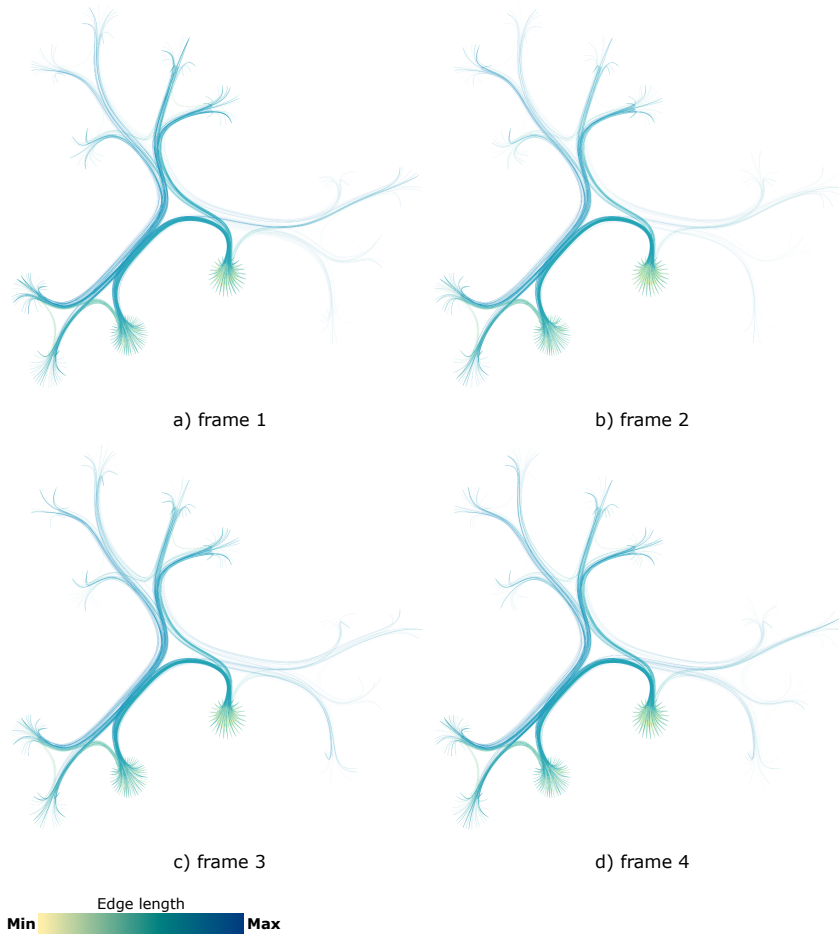d) frame 4

Edge length

Min ▢ **Max**

Figure 4.8: Four sample frames of the #NBABallot dynamic dataset, visualized with dynamic SDEB bundling.

terms of their *data*, while spatial proximity of the nodes will not encode that. As such, we get two conflicting signals in one image. In contrast, when the tree layout is driven by similarity (as explained earlier), both bundles between nodes and proximities of nodes encode the same information – similarity of nodes – and thereby lead to an easier to interpret image.

Visualizing similarities of datasets with SDEB is related in aims, but complementary in the used means, to the visualization of similarities proposed by the visual supertree (VST) techniques in Chapter 3. Both techniques offer a multi-scale, or level-of-detail, view of essentially the same data – similarities in a large set of observations. However, the VST visually encodes this multiscale by a number of straight-line tree drawings, where the different levels have significantly different numbers of nodes being drawn. At coarse levels, for instance, one visu-

alizes only supernodes, which are groups of similar observations. SDEB encodes this multiscale by the bundle structure, where the different levels have significantly different numbers of bundle branches being drawn. At coarse levels, for instance, one visualizes all leaf nodes (observations), but these are connected by a simplified bundle structure. As such, we can say that the VST is a *observation centric* visualization, whereas SDEB is a *relation centric* visualization. Of course, the borders between the two approaches are not hard: VST also shows relations in terms of edge-paths between nodes in the tree; and SDEB can also simplify visualizations by collapsing groups of similar nodes and replacing them by tag-cloud summarizations (Sec. 4.4.1).

SDEB depends, just as VST, on a reliable and efficient computation of a multilevel similarity tree. In both cases, such a tree is computed by using a combination of data clustering and the NJ algorithm (see Secs. 3.3 and 4.2.1). In both cases, the key parameters to set here are the number of clusters to create per level (parameter k if k-means clustering is used) and the maximum size of a cluster s_{max}. These parameters affect both types of visualizations, and finding their optimal values require experimentation with the concrete datasets at hand. Heuristics for determining good presets for these parameters can be offered here, as described both in this Chapter and in Chapter 3; however, even in this case, changing these parameters will influence the resulting visualization, so one cannot completely factor our a certain amount of user experimentation with parameter values to obtain a visualization deemed suitable for a given dataset and analysis task at hand.

4.6   Conclusion

In this chapter, we have presented a new technique for the simplified similarity-based visualization of large datasets. Our technique, called *Similarity-driven Edge Bundling (SDEB)*, uses a phylogenetic tree to guide the bending and grouping of edges connecting data entities. The proposed visualization is related to graph bundling in the sense of using the same visual metaphor – edge bundles – to reduce clutter and simplify the drawing of large relational datasets; and is in the same time different from general-purpose graph bundling in the sense of requiring as input a set of observations with a similarity matrix defined atop of them, providing thus a way to visualize the main edge patterns guided by a similarity backbone. As another contrast to existing general-purpose graph bundling techniques, we provide a multiscale way to explore the graph structure induced by similarity relations, where this multiscale is induced by a tree that represents the similarity data in a hierarchical fashion. This allows producing coarse-to-fine-grained views of a given dataset, and also to locally refine or coarsen the visualization, based on user input. We demonstrate our proposed SDEB method on several static (time-independent) and also one dynamic (time-dependent) dataset. Compared to clas-

sical state-of-the-art graph bundling techniques, we show that SDEB can achieve less clutter, and a better encoding of the structures present in datasets.

Many extensions are possible to SDEB. A quite interesting one is to generalize it to handle different types of node positioning. Rather than using the positioning of nodes determined by the layout of the similarity tree, we could, for instance, consider positioning leaf nodes based on their actual similarities, using a multidimensional projection method; or alternatively using fixed positions, such as in geographical datasets. Next, a similarity tree based on the attributes of these nodes could be constructed and fitted atop of these node positions, followed by the bundling proper. This would combine the advantages of the data-similarity-based bundling offered by SDEB with the increased flexibility of placing nodes based on other criteria. Another equally interesting extension would consider similarities of relations (edges) themselves, aside similarities of observations. The bundling could then be influenced by similarities of both nodes *and* edges, leading to a fully general method for bundled visualization of multivariate *and* relational datasets.

5

Attribute-based Explanation of Projections

In chapters 3 and 4, we have presented two different methods for explaining datasets in terms of the similarity of their entities. As explained within that context, such methods can be used also for multidimensional datasets. In that case, the similarity being used is based on the distance between multidimensional entities. Different ways to compute such distances have been introduced in Sec. 2.1.2.

The main advantage of the similarity-based visualizations in Chapters 3 and 4 is their ability to produce a multiscale representation of the similarity, which allows exploring large datasets in a compact way. This multiscale is given by the NJ similarity tree (Sec. 3.1). However, the similarity tree also introduces a disadvantage, as it forces nodes to be placed at specific spatial positions which may not optimally reflect similarity. In other words, while the tree *structure* captures similarity well, the *positions* of the tree leaves do that less. This may cause interpretation problems, as the spatial proximity of items is a strong cue to them being thought of as associated, or in our case, similar [191, 13].

Multidimensional *projections* take a complementary approach to similarity trees. They place observations in the visual (embedding) space so as their relative positions encode the observations' similarities [60, 171]. As such, inter-observation distance in the embedding space is a direct cue that maps similarity. Additionally, modern projection techniques have become increasingly more robust, computationally scalable, and easy to use [95, 135]. Yet, MPs have a fundamental limitation: While they show groups of similar entities, they cannot directly explain *why* these entities are similar [164, 28]. For similarity trees, such local explanations are possible, *e.g.* in the form of summarizations added by means of colors, textures, and tag clouds (see Secs. 3.2.3 and 4.4.1).

In this chapter, we propose a visual encoding that addresses precisely the above limitation of projections. Our encoding augments projection scatterplots with several explanatory aids that bring back in the dimension-related information that has disappeared during the projection. For this, we automatically and implicitly partition the projection space into zones of close (thus similar) observations and next compute an explanation of each zone based on the values of the data dimensions over its observations. We next use an image-based technique to construct a smooth-varying map that explains the entire projection by color coding. We enhance this explanation by explicit partitioning of the projection into large same-explanation zones which we annotate with dimension names and dimension values, much like tag clouds techniques used in earlier related work [143]. We demonstrate our explanation techniques by exploring several real-world multidimensional datasets from different application areas.

The structure of this chapter is as follows. Section 5.1 overviews related work in explaining multidimensional projection visualizations. Section 5.2 introduces our visual explanatory tools. Section 5.4 shows how our explanatory techniques can be used to understand the structure of high-dimensional data depicted as projections for several real-world datsets. Section 5.5 discusses our techniques. Section 5.6 concludes the chapter.

5.1   Related Work

For a dataset $D^n = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\} \subset \mathbb{R}^n$ of N n-dimensional points $\mathbf{p}_i = (\mathbf{p}_i^1, \ldots \mathbf{p}_i^n)$, projections create a dataset $D^m = \{\mathbf{q}_1, \ldots, \mathbf{q}_N\} \subset \mathbb{R}^m$, with $m \ll n$ (usually, $m \in \{2, 3\}$). If the projection function $P(\mathbf{p}_i) = \mathbf{q}_i$ preserves inter-point distances (*e.g.*, [95]) or nearest-neighbors (*e.g.*, [193]) when mapping D^n to D^m, a scatterplot of D^m can be used to reason about the D^n data structure.

As explained earlier in Sec. 2.4.4, visualizing a 'raw' projection as a point cloud of D^m shows similarity-related data patterns, but does not explain these in terms of the n dimensions of D^n, and is thus of limited use by itself. Hence, several methods have been proposed to *explain* projections. These methods have been overviewed in Sec. 2.4.4. We revisit this survey of explanatory methods for projections here from the perspective of our goal – explaining what makes groups of points similar in a projection. This way, we are able to highlight the strong points of such methods, which we aim to preserve, and also their weak points, which we aim to eliminate.

Interactive techniques explain MPs by showing on-demand information to help making sense of the D^m structures. Basic techniques include brushing close points in D^m to see which dimensions make them similar; scagnostics methods which pre-analyze all scatterplots in a scatterplot-matrix (SPLOM) [10] to detect which ones best capture interesting patterns in D^n [192]; and adding manual annotations to highlight specific items [54]. Overall, all such interactive methods are very precise, as they show actual dimension names and values for the explanation. However, they do not scale well visually (we cannot add annotations everywhere in a projection), and require manual effort.

Color coding: This well-known method colors all projected points with the value of a user-chosen dimension. This explains very well patterns such as minima, maxima, constant-value zones, and regions of high variation of the respective dimension. An attractive aspect of color coding is also that it is very scalable visually (for one dimension) and easy to interpret. However, this technique can show a single dimension at a time, and showing multiple dimensions in turn overloads easily the user's memory. We note that color coding is also used to explain projection errors [160, 135]. Various interpolation methods have been proposed in this context to 'fill in' the gaps in a projection scatterplot. This replaces the harder task of

interpreting a color-coded scatterplot with the easier task of interpreting a map having differently colored regions [118].

Axis techniques explain the relations of the dimensions of the original space D^n with those of the projection space D^m. Such techniques include biplots, which show where the original D^n dimensions, or axes, project in D^m [71, 70]; and histograms showing how the m dimensions of D^m are composed of (linear) combinations of the D^n dimensions [18, 28]. However, such methods are global in nature. Moreover, they work less well for nonlinear projections (*e.g.*, [193, 95]), which require different *local* explanations of neighborhoods in D^m.

Clustering organizes the D^m points into closely-related groups, which may admit simple explanations. The general idea is to segment the projection space into groups of closely-spaced points, since these groups will arguably be those that a user perceives first, and wants to have explained. Next, various statistics are computed on each group, to find out which dimensions and/or dimension values best explain the closeness of its points. Alternatively, representative observations are selected from a cluster and their details are displayed to explain the cluster. Finally, the results of this analysis are displayed using annotations such as labels and tag clouds [180, 131, 99, 143]. These methods are visually quite scalable, offer a relatively good level of detail, and are intuitive. However, clustering always introduces some arbitrary decisions on where to draw cluster borders, and can be sensitive to various parameters (see also Sec. 4.5). Also, explaining a cluster by a single representative can be confusing in cases when individual observations do not carry particular strong meanings. In such cases, one would like to explain D^m by dimensions or dimension-values rather than observations.

5.2   Local attribute-based explanation

From the review of existing explanatory methods for projections given in Sec. 5.1, we see that no single such method meets all desired qualities, *i.e.*, local explanation, sufficient level of detail, visual scalability, ease of interpretation, and automated use. Different methods have, however, several strong points. Specifically, we see that

- *color coding* methods are intuitive and easy to interpret;

- *annotation* based methods are very precise in their explanations;

- *clustering* methods achieve a high degree of summarization.

As such, we propose in the remainder of this chapter a novel explanation method which aims to combine the above-mentioned advantages. This will be done by adapting and extending the techniques that the aforementioned explanatory methods use to achieve their goals. We call this method a *local attribute-based explanation*, as the method explains different zones in a projection differently (as opposed

to axis-based methods) and the explanation is based on attributes, or dimensions, and their values over the dataset. This method is explained next.

Virtually all projection techniques aim to place points which are similar in D^n closely in D^m. Of course, certain methods may achieve better or worse results in this placement than other methods. However, as explained in Sec. 2.4.3, many established methods exist for assessing the precision of a projection, or its ability to preserve distances. We assume in the next discussions that the projections we study have sufficient precision for the tasks at hand, and that this precision has been already verified by means of the error metrics mentioned earlier. We also assume that the resulting projections are two-dimensional, *i.e.*, $m = 2$. This covers the majority of projection usages known in the literature.

The key patterns that one sees, and needs to explain, in a 2D projection scatterplot are groups (closely placed points) and outliers (points far away from the rest of the scatterplot). As such, our approach aims to explain why close points in D^m are similar. For this, we define, for each $\mathbf{q}_i \in D^m$, a 2D neighborhood $v_i^P = \{\mathbf{q} \in D^m | \|\mathbf{q} - \mathbf{q}_i\| \leqslant \rho\}$ as all projected points closer to $\mathbf{q}_i$ than a given radius ρ. This induces a neighborhood $v_i = P^{-1}(v_i^P) \subset D^n$ of $\mathbf{p}_i$, over which a ranking $\mu_i = (\mu_i^1, \dots \mu_i^n) \in \mathbb{R}^n$ for all n dimensions of $\mathbf{p}_i$ is computed. The lower a rank μ_i^j is, the better can dimension j explain the similarity of points over v_i. Sections 5.2.1 and 5.2.2 next outline the rank computation and visual rank encoding. Section 5.3 next presents a different way to explain projections based on clusters.

5.2.1 Dimension ranking

To compute ranks of all n dimensions over a neighborhood of points, we intuitively want to find out how much each of the dimensions contributes to the fact that the respective points have been placed close to each other in the projection space. To do this, we study next how the n-dimensional distance between two points is affected by its individual components caused by all the n dimensions.

Euclidean similarity ranking:

To compute the ranks μ_i, a first way is to consider the Euclidean distance $d(\mathbf{p}, \mathbf{r}) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$. Let $lc_{\mathbf{p},\mathbf{r}}^j$ be the contribution of dimension j to the distance between two D^n points $\mathbf{p}$ and $\mathbf{r}$, defined as

$$lc_{\mathbf{p},\mathbf{r}}^j = \frac{(\mathbf{p}^j - \mathbf{r}^j)^2}{\|\mathbf{p} - \mathbf{r}\|^2}. \tag{5.1}$$

For each $\mathbf{p}_i \in D^n$, the local contribution of dimension j is next defined as the average of the distance-contributions between $\mathbf{p}_i$ and its neighbors $\mathbf{r} \in v_i$, where $v_i \subset D^n$ is the image of a small neighborhood vu_i^P located in the projection

D^m by the inverse P^{-1} of the projection function. Hence, we compute this local contribution of dimension j over v_i as

$$\overline{lc_i^j} = \frac{\sum_{\mathbf{r} \in v_i} lc_{\mathbf{p}_i,\mathbf{r}}^j}{|v_i|}. \tag{5.2}$$

Our key idea next is to explain a neighborhood v^P by highlighting dimensions that contribute to similarities in v *and* that are not similar outside v – *i.e.*, dimensions that discriminate between points inside and outside v^P. For this, the global contributions (gc^j) of al dimensions j is first computed, by applying Eqn. 5.2 to the centroid of the whole dataset D^n and defining the neighborhood v of this point as the entire set D^n. The final Euclidean dimension-ranks are then computed, for each point j, as ratios between local and global contributions, normalized to capture the relative importance of different dimensions. Thus, the rank of dimension j for point i is

$$\mu_i^j = \frac{\overline{lc_i^j}/gc^j}{\sum_{k=1}^n \left(\overline{lc_i^k}/gc^k\right)}. \tag{5.3}$$

Variance similarity ranking: We also investigated an alternative to the Euclidean ranking, by using data variances. Let $GV = (var(\mathbf{p}^1), \ldots, var(\mathbf{p}^n))$ be the global variance of all dimensions over D^n. Here, $var(\mathbf{p}^j)$, denoted next by GV^j, is the variance of the jth dimension of the data points over the entire dataset. To capture how dimension j contributes to similarity over a neighborhoods v_i centered at point i, the ratio of the local variance LV_i^j over v_i and global variance GV^j is computed, normalized to indicate relative importance of dimensions. Here, LV_i^j is simply the variance of dimension j over the data points in v_i. This gives the rank of dimension j for point i as

$$\mu_i^j = \frac{LV_i^j/GV^j}{\sum_{j=1}^n (LV_i^j/GV^j)}. \tag{5.4}$$

Note the similarity of Eqn. 5.4 to Eqn. 5.3.

Dissimilarity ranking: A projection may also be influenced by dissimilar values among its points. Indeed, two point groups might be projected far away from each other in D^m even when sharing similar values for some dimensions, if the remaining dimensions are sufficiently different. To explain such differences, we propose a *dissimilarity ranking* that finds dimensions that contribute most to dissimilarity between two selected point groups in D^m. For this, the user first manually selects two point-groups Q and Q' which are, typically, far away in D^m and whose dissimilarity she wants to explain. Let P and P' be the counterparts in D^n of Q and Q' respectively. Given two points $\mathbf{q} \in Q$ and $\mathbf{q}' \in Q'$ with high-dimensional coun-

terparts $\mathbf{p} \in P$ and $\mathbf{p}' \in P'$, we compute using Eqn. 5.2 the contributions $lc_{\mathbf{p},\mathbf{p}'}^{j}$ of all dimensions j to the Euclidean distance between $\mathbf{p}$ and $\mathbf{p}'$. We next define the contribution of dimension j to the distance between $\mathbf{p}$ and the *entire* group P' as the average

$$lc_{\mathbf{p},P'}^{j} = \frac{\sum_{\mathbf{p}' \in P'} lc_{\mathbf{p},\mathbf{p}'}^{j}}{|P'|}. \tag{5.5}$$

For a point $\mathbf{p}_i \in P$, let μ_i^j be the jth largest value of $lc_{\mathbf{p},P'}^{1}, \ldots, lc_{\mathbf{p},P'}^{n}$. The ranking vector $\mu_i = (\mu_i^1, \ldots, \mu_i^n)$, which we construct as output, thus gives the most-to-least important dimensions that contribute to the dissimilarity between point $\mathbf{p}_i$ and all points in P'. The dissimilarities of points $\mathbf{p}' \in P'$ with respect to the entire group P are computed analogously.

Note that both the Euclidean and variance rankings (Eqns. 5.3 and 5.4) are *similarity* rankings – that is, low values of μ_i^j indicate dimensions j which are better for explaining a local neighborhood. Indeed, a low rank indicates more similar values for that dimension, *i.e.* a stronger cohesion of points from the perspective of the property sampled by that particular attribute. In contrast, the dissimilarity ranking implied by Eqn. 5.5 has an opposite interpretation –high values of μ_i^j indicate dimensions j which are better for explaining why two point-groups are different. Both above effects will be illustrated in Sec. 5.4.

A second note relates to the analogy of our rankings to the local projection errors proposed by Martins *et al.* [118] (see also Sec. 2.4.3). Our similarity rankings are analogous to the false neighbors metric of Martins *et al.* – that is, they characterize groups of *close* points in a projection. In contrast, our dissimilarity ranking is analogous to the missing neighbors metric of Martins *et al.* – that is, it characterizes groups of *far away* points in a projection. This analogy also extends to the input we need to compute our rankings: For the similarity rankings, all we need is the size of a neighborhood; given this, we (and also Martins *et al.*) can compute the respective metrics at any point in a projection. Moreover, such metrics can be easily visualized for *all* points in a projection, since a point implicitly defines its surrounding neighborhood. In contrast, for the dissimilarity ranking, we need an explicit *selection* of two groups of (far-away) points that we want to know why they are dissimilar. Analogously, Martns *et al.* needs to explicitly select the point(s) for which missing neighbors are to be computed. Visualizing such dissimilarity metrics also cannot be done for all points in a single image, since, in our case, that would involve selecting all pairs of far-away groups, and in the case of Martins *et al.* it would involve showing all missing neighbors, potentially spread anywhere over the projection, for each projected point. These analogies will influence our visual encoding of projection explanations, described in the next section.

A final note regards the choice between the Euclidean and variance-based similarity rankings. We have experimented with both rankings by applying them to create explanatory visualizations of a variety of datasets of different num-

bers of observations and dimensions. In general, we have seen that the variance-based ranking is more robust, *i.e.*, it produces locally less sharply varying top-explanation dimensions for neighborhoods of the data where we know that the n-dimensional distribution of observations does not rapidly change. As such, we shall use the variance-based similarity ranking (Eqn. 5.4 in the remainder of this chapter, unless explicitly stated otherwise.

5.2.2 Visual encoding

Basic idea: For each point i of the dataset, we compute a vector $\{(j, \mu_i^j)\}_{1 \leqslant j \leqslant n}$ with the IDs and ranks of all its n dimensions, sorted on rank values increasingly (for the Euclidean and variance rankings) or decreasingly (for the dissimilarity ranking). Next, we select the C dimensions having top-ranks for most of the N points in our dataset, and map their IDs to colors via a categorical colormap with $C = 9$ entries, built using ColorBrewer[1]. This way, dimensions which are top-rank for many points get mapped to distinct colors. Dimensions which are top-rank for few points do not get colors (due to the colormap's limited size C) and are mapped to the reserved color dark blue.

Encoding confidence: As noted previously, a top-rank dimension is important, but not *solely* responsible for the similarity of a projected point $\mathbf{q}_i$ with its neighbors. The same is true for the dissimilarity of points. To show this, we analyze the top-ranks of points in a 2D neighborhood ν_c^P centered at $\mathbf{q}_i$, defined like ν^P but using a smaller radius $\rho_c < \rho$. In detail: Let t be the ID of the top-rank dimension for $\mathbf{q}_i$, *i.e.*, $t = \arg\max_{1 \leqslant j \leqslant n} \mu_i^j$. The confidence c_i^t of t being the top-rank dimension that best explains the similarity of $\mathbf{q}_i$ with its neighbors is next computed as the sum of ranks μ_j^t for all points $\mathbf{q}_j \in \nu_c^P$ having t as top-rank dimension, normalized by the sum of all top-ranks over all points in ν_c^P, *i.e.*

$$c_i^t = \frac{\sum_{\mathbf{q}_j \in \nu_c^P \wedge \arg\max_k \mu_j^k = t} \mu_j^t}{\sum_{\mathbf{q}_j \in \nu_c^P} \max_k \mu_j^k}. \tag{5.6}$$

Intuitively, the above operation acts as a smoothing filter with kernel radius ρ_c that assigns high confidence to homogeneous (same top-rank) regions and low confidence to mixed regions (having points with different top-ranks). High ρ_c values de-emphasize outliers (points having different top-ranks than their neighbors), while low ρ_c values emphasize the variation of ranking confidence over finer scales (see also Fig. 5.1 discussed next).

Visualization: The top-ranks and confidences of projected points are encoded into point hues and brightnesses respectively, using the dense map technique of Mar-

---

1 http://www.colorbrewer.org
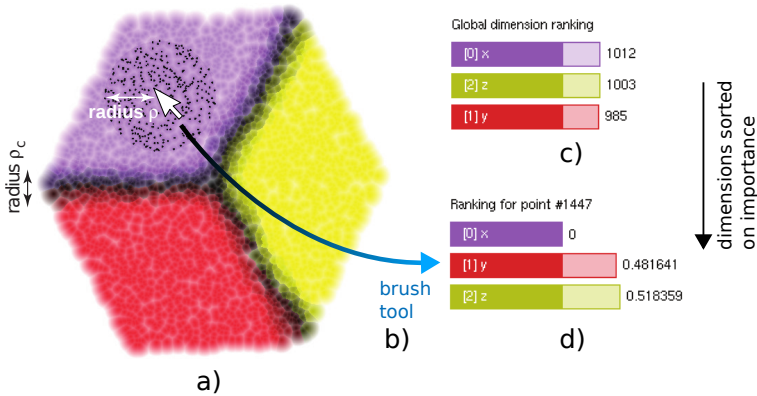
Figure 5.1: Visual explanation of a synthetic cube dataset.

tins *et al.* [118], *i.e.*, nearest-neighbor (Voronoi) interpolation of the scatterplot, done by drawing textured and colored Gaussian splats centered at the projected points.

Figure 5.1a shows this for a simple synthetic dataset of 3000 points randomly sampled from three faces of a cube, and next perturbed by uniform spatial random noise of amplitude equal to 5% of the dataset's extent. We project this 3D dataset to 2D using PCA [96] (since PCA is a very well known technique and preserves distances well for this very simple dataset) and ranked the dimensions by the variance metric. The radius parameter ρ is set to 10% of the diameter of D^m. The resulting explanation (Fig. 5.1) shows that the projection consists of three 'zones' that correspond very well to cube's faces, each being very well explained by a single dimension, as expected. Points close to face intersections are darker, so their explanation by a *single* dimension is less confident, as expected. Indeed, if we center our neighborhood close to an edge, it will include points from at least two faces of the cube, thus points whose similarity cannot be explained by a single dimension.

We add next a global ranking legend (Fig. 5.1c) to show which color encodes the identity which dimension, and how many points have that dimension as top rank. The last value also sorts the bars in the global ranking legend top-down, so one can easily see the *overall* importance of a given dimension for explaining the similarity of all the points in the input dataset. In our case, the legend shows that the point count is divided in three roughly equal parts, which is correct, given the roughly equal number of samples on the three cube faces.

A brush tool is provided to interactively inspect the ranks μ_i^j for a given point i. These are shown in a second bar chart (Fig. 5.1d). Here, dimensions are sorted top-down in the same order as in the global ranking legend (Fig. 5.1c), so one can see how important are each dimensions *locally* as opposed to globally. For our cube example, we see that the top-rank dimension x (purple) has variance 0, which is indeed correct, as the brushed point is in the middle of a face orthogonal

to the x axis. Dimensions y and z have low, and roughly equal, variances. These are correct, since our points, sampled from the cube faces, are perturbed by a small amount of noise, as explained earlier.

Multiple-dimension explanation: If the top k ranks μ_i^j of a point j are very similar, the 'winning' top dimension $\arg\max_j \mu_i^j$ may be subject to noise. As such, we propose to explain such areas using multiple (top-ranked) dimensions. Given a point i, we first define its *top-rank set* as all the top-ranked dimensions j whose ranks μ_i^j sum up to be (just) larger than a user-defined small threshold τ. This set contains thus all to–ranked dimensions whose cumulative effect on the ranking metric is lower than τ. If the standard deviation of ranks μ_i^j is small, then this set will be large, *i.e.*, we need many dimensions to explain the neighborhood around point i. In the opposite case, this set will be small – in the limit, it contains a single element $\max_j \mu_i^j$, so the top-rank set becomes identical to the top-rank discussed earlier. To visualize top-rank sets, we assign categorical colors to the C most-frequent rank-sets in the projection, and map the remaining sets by the reserved color dark blue. Examples hereof are shown next in Sec. 5.2.3, where we also discuss several examples.

5.2.3   Example Applications

We next use our method to explain projections from three real datasets. As projection P, we used LAMP [95] due to its accuracy and computational speed, both studied extensively in [118]. As outlined at the end of Sec. 5.2.1, we use the variance ranking. As parameter values we used $\rho = 10\%$ for the projection diameter (largest distance between any two points) and $\tau = 0.05$. Dimension labels were added manually on the projection to help easier identification.

We next illustrate the use of our local attribute-based explanations of projections for three datasets: quality of wines, quality of software projects, and US counties.

Wine quality: This dataset has 6497 samples of Portuguese *vinho verde* wine (4898 red wine; 1599 white wine) [33]. Each sample has $n = 12$ physicochemical measures like acidity, residual sugar, and alcohol rate. The projection is shown in Fig. 5.2 a). If we imagine the visualization of this projection without the attribute explanations in the figure, we would see an unstructured single clump of points, which cannot be further easily interpreted or locally analyzed, since there are no outlier clusters.

We start exploring this dataset using first our single-dimension explanation. This explanation splits the projection clump into three regions defined by the top-rank dimensions *alcohol rate*, *sodium chloride/dm^3* and *residual sugar*, and a smaller group defined by *volatile acidity* (Fig. 5.2 a). Zones close to region borders are dark, intuitively showing that they cannot be explained by a single dimension.
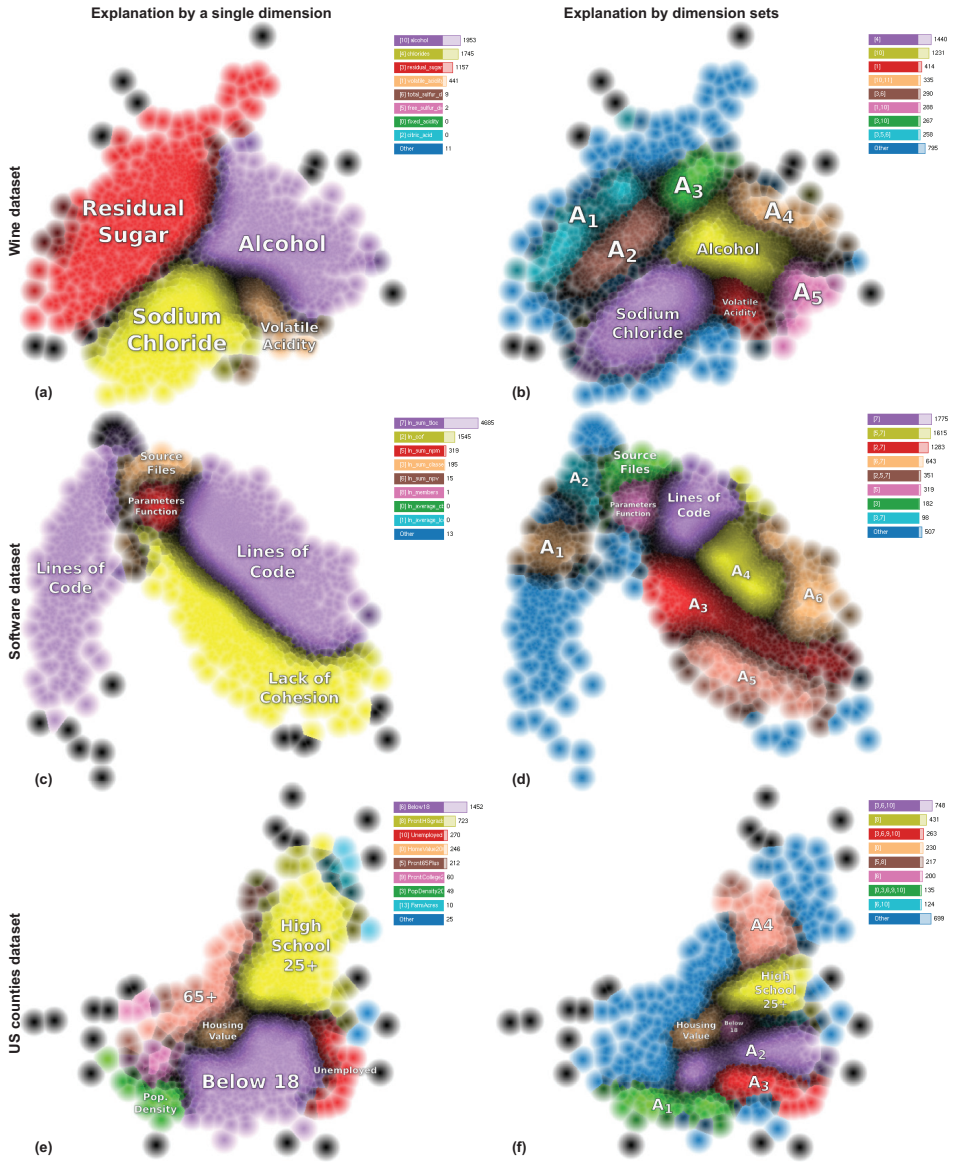
Figure 5.2: Visual explanations of three datasets using a single dimension (left column) and dimension-sets (right column).

Using the brush tool, we discover that the first two dimensions account for 5% of the total rankings on several areas.

To get more insight into what makes points in a region similar, we use next the dimension-set explanation on the same dataset and projection. The dimension-set explanation (Fig. 5.2 b) splits the regions identified by the single-dimension explanation into finer detail. First, *residual sugar* is split into two subregions A_1 and A_2. A_1 also include the dimensions *free sulfur dioxide* and *total sulfur dioxide* in its explanation, and A_2 also includes *total sulfur dioxide*. Hence, sulfur dioxide is closely related to residual sugar to explain these regions. Region A_3 appears in the border of two regions of the previous map, and is defined by the union of these dimensions. In subregion A_4, the dimension *wine quality* was added to the explanation, showing samples with similar quality and alcohol values. Subregion A_5 covers the union of the former *alcohol* and *volatile acidity* regions. Other subregions remain best explained by the same top-ranked dimensions since, over them, the sum of ranks between the first and second top-rank dimensions is above the threshold τ. Finally, about 12% of the points are explained by less-frequent dimension sets, mapped by the color dark blue.

Quality of software projects: Our second dataset describes 6773 software projects from *sourceforge.net* written in C [121]. Each project has 12 dimensions (11 software quality metrics and the project's total download count). These metrics were extracted using static analysis tools. The original intention of this study, presented in [121], was to find out whether correlations exist from these objective measurements of software quality, and the number of downloads of each software project. In our study of this dataset, we only use the software quality metrics.

The projection of this dataset shows two large connected regions (Fig. 5.2 c,d). Single-dimension explanation (Fig. 5.2 c) shows that the left region is best explained by dimension *total lines of code*. The right region is roughly evenly split into two parts explained by dimensions *total lines of code* and *lack of function cohesion* respectively. Several small groups and a low-confidence border connect the above two regions. Dimension-set explanation shows that most subregions can be explained by two dimensions (Fig. 5.2 d). The left region becomes now mainly blue, showing that there are too many small-scale regions that would need *more* than one dimension to be explained, and that no such region has a sufficient number of points to 'win' a place in our limited categorical colormap. Exceptions are the subregions A_1, which adds the quality metric *number of public variables*, and A_2, which adds the metric *number of source files*, which is also related to the neighbor green region. The right region is split in several compact sub-regions: A_3 is a union between *lines of code* and *lack of function cohesion*; A_5 adds the same dimension of A_3 and also the *number of function parameters*; A_4 also adds the number of function parameters; finally, A_6 adds the metric *number of public variables* to its explanation. As before, we see how the confidence of the explanation smoothly drops from the center of a region towards its border, as shown by the luminance

variation.

US counties: This 12-dimensional dataset describes social, economic, and environmental data from 3138 USA cities [133]. Its projection yields a single visual cluster (Fig. 5.2f,g). Single-dimension explanation shows six main regions, chiefly explained by dimensions related to social statistics (Fig. 5.2e). The dimension-set explanation (Fig. 5.2f) splits these regions, as follows: The former *below 18* region gets split into four. One subregion (A_2) remains best explained by the *below 18* dimension. A_2 is explained by the *unemployed* and *population density* dimensions which also defined the two neighbor regions in the single-dimension explanation. A_3 is explained by the same dimensions, plus the dimension *percent of college/higher graduates*. Hence, A_3 can be seen as a more specific subset of A_2. Finally, A_1 is defined by the same dimensions as A_3, plus the dimension *median of owner-occupied housing value*, being thus an even more specialized subset of A_2. The subregion A_4 is defined by dimensions *percent of high school graduates age 25+* and *population $\geqslant$ 65 years old*. Finally, the region defined by *median of owner-occupied housing value* stayed the same as the single-dimension explanation map, indicating that this dimension is sufficient to clearly define this region.

5.2.4   Parameters Discussion

Our basic visual encoding has three main parameters which are intuitive and simple to control:

- ρ acts as a scale factor – small values create more detailed explanations and thinner region borders (Fig. 5.3a) , but also emphasize outliers more. Larger values create less regions but thicker fuzzy borders, thus a coarse scale explanation (Fig. 5.3b);

- ρ_c acts as a smoothing filter: small values create more noisy regions but thinner borders (Fig. 5.3c); large values create smooth regions but thicker borders (Fig. 5.3d);

- τ controls the coherence of points in a region: small values create less-coherent regions (Fig. 5.3e); large values create many strongly-coherent regions (Fig. 5.3f).

Once values for these parameters are set, the partition of a projection into regions, which are next explained by means of color-coding is fully automatic. The only interaction required here is brushing regions to bring up the dimension-value bars or clicking on them to select groups of points for the dissimilarity ranking, if this is desired.
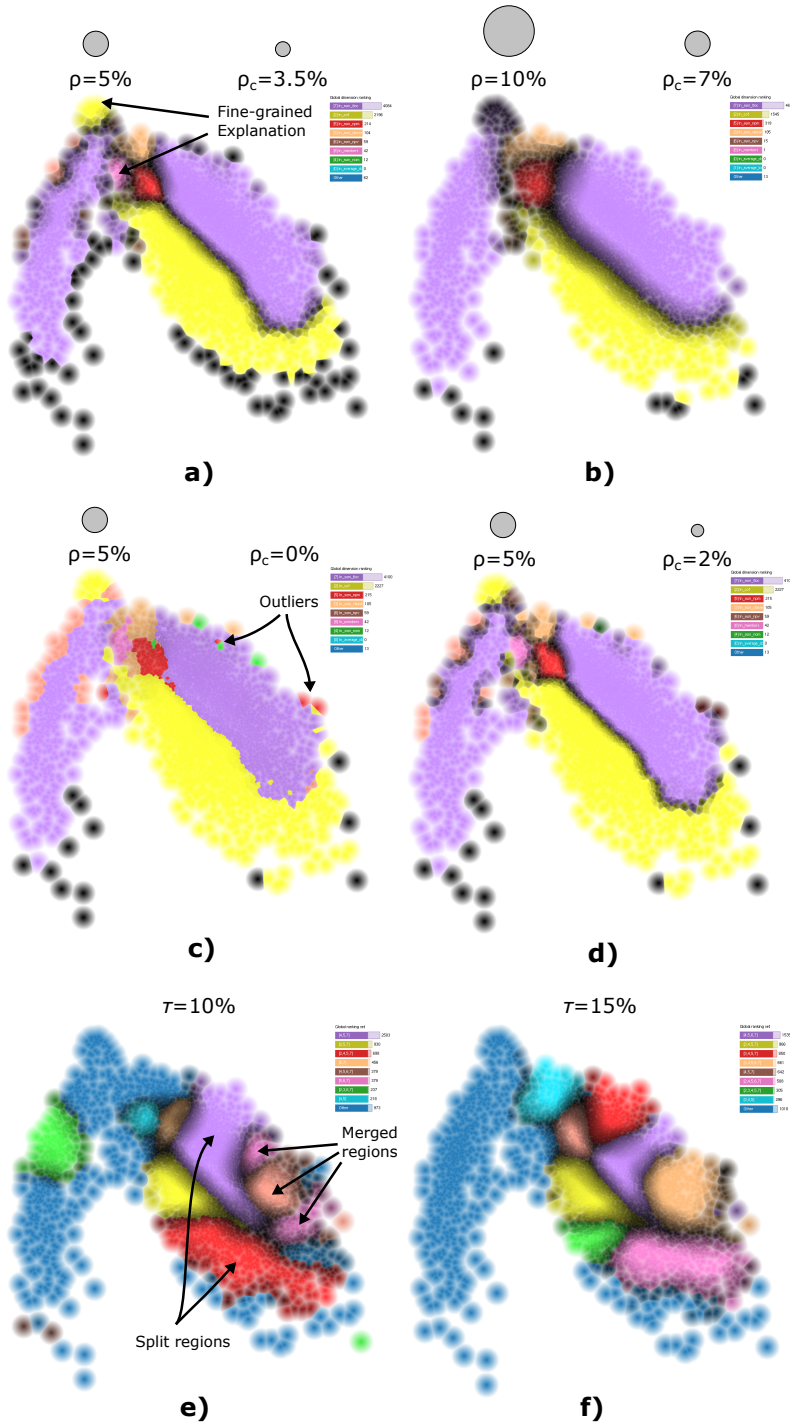
Figure 5.3: Effect of parameters ρ, ρ$_c$, and τ on the visual encoding. Values of ρ and ρ$_c$ are given in percentages of the size (diameter of circumscribing circle) of the 2D projection. Values of τ are pecentages of the similarity ranking metric μ.

5.3 Improved Visual Encoding

The visual explanations in Sec. 5.2 produce a hue-saturation map that locally encodes the dimension(s) that best explain point similarity in the projection. This replaces the discrete scatterplot with an image in which compact color spots indicate dataset regions being similar for different reasons (dimensions). This gives good results for relatively simple projections having a small number of large spots.

However, we noticed some important problems on that approach.

Firstly, the visual encoding based on nearest-neighbor interpolation of colors creates visual artifacts which ressemble a polygonal, mosaic-like, structure superimposed on the projection. These artifacts are especially visible in areas where different colors meet, and when the local density of the projected points is medium or low. Such artifacts are visible along the edges of the cube dataset projection in Fig. 5.1, and are repeated with a detail image in Fig. 5.4a, for clarity. Such artifacts are distracting and do not convey any additional information to the projection explanation.

Secondly, our color-based visual map can lead to interpretation problems, depending of the assigned color/brightness configuration of different regions. For example, a highly confident region, drawn using bright brown, might visually look similar to a low confident region, drawn as dark orange. This is a problem caused due to the lack of very different categorical colors in an already-large categorical colormap, and due to the fact that we modify these colors to encode confidence, by changing their brightness.

A third limitation of our visual encoding is the lack of information about the values of the explained dimensions. Figure 5.2 c is a good example of such limitation. The explanation of this projection shows two big regions, drawn in purple, explained by the same attribute *lines of code*. Since these regions are far apart, it is clear that observations contained by them have different values of this attribute. However, the explanation does not show which are these different values.

To alleviate these problems, we enhance the basic visual encoding described in Sec. 5.2.2. First, we use an improved interpolation of colors to create a smoother visualization and remove the visual artifacts caused by the Vornoi interpolation (Sec. 5.3.1). We next compute explicit same-explanation clusters, providing additional informational cues to understand these, by a four-step process: cluster identification, delineation, labeling, and dimension-value explanation (Secs. 5.3.2-5.3.5). These improvements are discussed next.

5.3.1 Smooth Explanatory Map

To explain this improvement, let us go back to the example that explained the 2D projection of three faces of a cube dataset (Sec. 5.2.2). The result of our visual encoding presented so far is displayed in Fig. 5.4, for clarity. Even for this simple dataset we can notice a distracting reticular pattern between the plot points

(Fig. 5.4a, insets). This pattern manifests itself by non-smooth variations in both brightness and hue. The borders between different-hue regions actually follow the Voronoi cells of the 2D scatterplot. The appearance of this pattern is not surprising, as the splatting technique we use, borrowed from [118], is simply a nearest-neighbor interpolation in visual space of the categorical information (dimension IDs) computed by the ranking on the projected points. As mentioned earlier, this pattern is distracting and does not convey any actual information.

We improve this issue by using Shepard interpolation for both hue and brightness (Fig. 5.4b), as follows. For each pixel $\mathbf{x}$ in the image, we find all projected points $\mathbf{q}_i$ in a small neighborhood $\nu_r(\mathbf{x})$ of radius r pixels centered at $\mathbf{x}$ (with $r = 5$ pixels set to all our experiments). Let $\phi : \mathbb{R} \to \mathbb{R}^+$ be a smooth monotonic decaying filter function – in our experiments, we chose $\phi(x) = \exp(-(\frac{x}{r})^2)$. We interpolate the dimension ranks μ_i^j of all $\mathbf{q}_i$ for all dimensions j at pixel $\mathbf{x}$ as

$$\mu(\mathbf{x})^j = \frac{\sum_{\mathbf{q}_i \in \nu_r(\mathbf{x})} \phi(\|\mathbf{q}_i - \mathbf{x}\|)\mu_i^j}{\sum_{\mathbf{q}_i \in \nu_r(\mathbf{x})} \phi(\|\mathbf{q}_i - \mathbf{x}\|)}, \tag{5.7}$$

and set the color of pixel $\mathbf{x}$ by finding the dimension that maximizes $\mu(\mathbf{x})^j$ for all $j \in \{1, \ldots, n\}$. The brightness of $\mathbf{x}$ is computed analogously, by using the interpolation in Eqn. 5.7 for the confidences c_i^j. In other words, we use a Shepard interpolation to find, *at each pixel*, the values of the ranks and confidences, based on the values of these signals over the data points around $\mathbf{x}$ in a small neighborhood ν_r, and next use these interpolated values to determine the actual color and brightness for the pixel. Note that using the continuous Shepard interpolation on the values of μ_i^j and c_i^j is sound, since these are continuous signals over $\mathbb{R}^2$, and the categorical color mapping is applied *after* interpolation. The alternative of interpolating the categorical colors would, of course, not make sense.
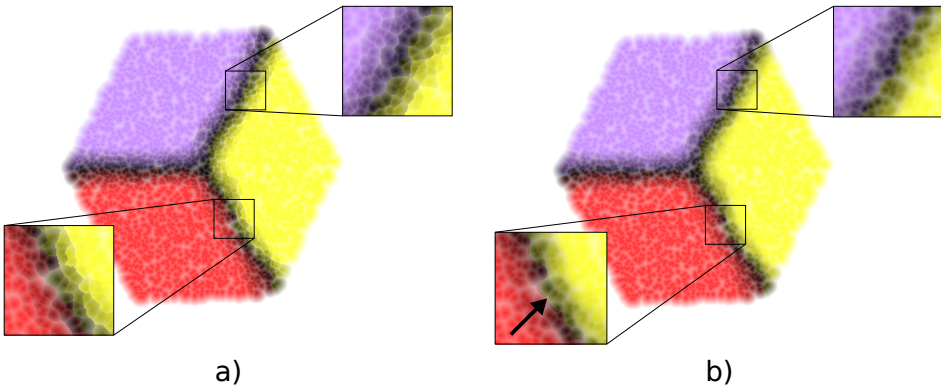
Figure 5.4: Visual explanation of a synthetic cube dataset: (a) original and (b) smooth.

Figure 5.4 shows the effects of the smooth Shepard interpolation for the cube dataset. As visible, the Voronoi-like artifacts have been removed. We can now see a clear *and* smooth border that separates the different hue regions. Performing the Shepard interpolation is computationally as efficient as the original nearest-neighbor splatting, since we implement Eqn. 5.7 using NVidia's CUDA platform as a simple 2D convolution. Practically, this allows us to render images like Fig. 5.4 for projections of thousands of observations in real-time on a consumer-grade PC availing of a CUDA-capable graphics card.

The visualization techniques explained so far produce a projection where points are *implicitly* grouped, by color-coding, on the dimension(s) best explaining their local similarity. While this explanation of a projection is automatically computed, which is attractive from an usability perspective, it also has some limitations: Reading the explanation of a group involves searching its color in the color legend, which can be tedious for datasets having tens of dimensions or time-dependent datasets. Moreover, as we have outlined above, changing brightnesses to encode confidence can make different categorical colors look similar. Finally, we cannot place explanatory labels atop of such groups (as shown in Fig. 5.2) automatically, since the groups are implicit.

We address the above issues by computing *explicit* same-explanation clusters, in four steps: cluster identification (Sec. 5.3.2), cluster delineation (Sec. 5.3.3), cluster labeling (Sec. 5.3.4), and dimension-value explanation (Sec. 5.3.5). These steps are explained next.

5.3.2   Cluster identification

Given a projection D^m where each point is explained by a top-rank dimension or dimension-set, we segment D^m into compact same-explanation zones, using a simple connected components approach. Here, a point in D^m is considered to be connected to its nearest neighbor having the same explanation *and* a confidence higher than 50%. This extracts from D^m a set of clusters D_i^m which, intuitively, contain same-hue points and meet at the dark (low-confidence) areas. For instance, the projection in Fig. 5.4 is split this way into three clusters (red, yellow, and purple regions). This process is fast to execute, simple to implement, and requires no user intervention or parameter setting, unlike *e.g.* hierarchical clustering used for similar tasks [131].

Note, however, that this segmentation does not produce a *partition* of D^m: very low-confident points in D^m will not be included in any cluster. This is desired, since we want next to explain such point clusters using explicit dimension labels; as such, the clusters should only contain high-confidence points, otherwise the label explanation may be misleading. Separately, the connectivity criterion is used to ensure that the resulting clusters are compact; this will help us when positioning labels to explain them, as described next.

5.3.3   Cluster delineation

The second step in our cluster-based explanation is to *delineate* the clusters D_i^m created by the segmentation procedure presented in Sec. 5.3.2. For each cluster D_i^m, delineation aims to build a closed, connected, self-intersection-free, and smooth outline $L(D_i^m) \subset \mathbb{R}^2$ that completely surrounds the projected points in D_i^m and also wraps tightly around these points. The purpose of these outlines is threefold: They (a) make the regions which are next explained by labels explicit in the visualization; (b) allow placing the labels automatically; and (c) allow users to select all points in a cluster easily, by a single click operation inside the outline.
  Delineation is implemented as a two step process, as follows.

Inflation: We first compute the convex hull $H(D_i^m)$ of all points in a cluster. This is a convex polygon that tightly surrounds all points in D_i^m. For this, we use the convex hull implementation which is part of the well-known Triangle library [166], which is easy to use, fast, and very robust. Other convex hull implementations can be, of course, used instead. Next, we slightly inflate (upscale) H with a small uniform scaling factor, roughly 5% of its size, with respect to its barycenter. This creates a small offset between the hull and the surrounding points, which will help the shrinking procedure described next. Finally, we sample H is uniformly in arc-length space, which transforms it into a finely-sampled closed 2D contour polyline $L(D_i^m)$.

Shrinking: L is next iteratively shrunk by moving its points with a small step along the contour's inward normal. We alternate shrinking iterations with Laplacian smoothing iterations, following the idea of mean curvature flow [26] or gradient vector flow [208]. This way, the contour's curvature does not increase; this would be undesirable given that we want smooth delineation and, also, high curvature may cause normal estimation problems. After each shrink-smooth pass (about 10 in total), L is resampled again so as to preserve an uniform point density during the shrinking. Contour points are not moved if they get closer than a user-given offset value α to a point in D_i^m or if motion would cause L to self-intersect. The obtained outline is visualized by drawing it in black.
  Figure 5.5 shows two cluster outlines built for two different offset values α by the above process. As visible, small α values create a tighter, but more tortuous, outline; larger α values create a smoother, but looser, outline. This outlining technique is fully automatic; handles any cluster configurations (convex, concave, compact, or variable density); produces by construction outlines that surround all points, have a given smoothness, are compact, and are intersection-free; and has a single user parameter (α) which is simple and robust to set. Arguably the best know related technique is alpha shapes [51] which can also produce concave contours surrounding all given points, but does not guarantee contour smoothness or user-prescribed offsets. Other related techniques include, most notably, the algorithm of Byelas *et al.* [21], which also uses iterative shrinking of a convex
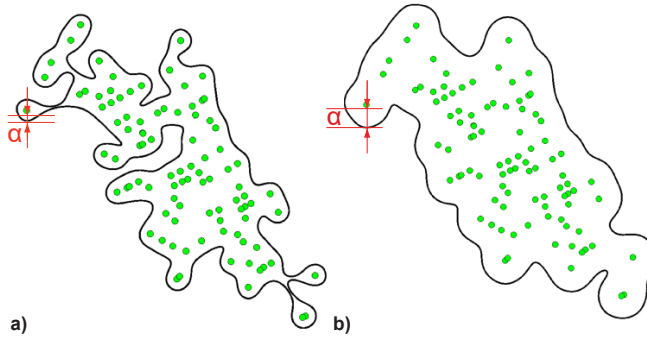
Figure 5.5: Outlines computed for a point cluster for (a) small and (b) large offsets α.

hull to compute outlines of groups of elements in UML class diagrams. Our technique can be seen as a simplified version of the above algorithm, which includes additional (complex) steps to handle input configurations which consist of sets of rectangles (rather than points) which are possibly far from each other in the drawing. Another difference with Byelas *et al.* regards the precision of outlines. In our case, we have observed that smoother and looser outlines are easier to interpret, in the sense of quickly locating the elements they contain when visualizing a projection. The potential imperfection in terms of tight delineation of the points in a cluster caused by loose outlines does not appear to affect the assessment of the visualization. This is in contrast to Byelas *et al.*, where outlines need to be very precise in terms of potential overlaps and intersections. This is explained by the different goals of the two types of outlines. Byelas *et al.* use these to reason about the precise containment of a graphical element in a set of outlined groups, which can have any sizes and spatial distributions of elements over the 2D surface. In our case, we use outlines only as indications of large and compact groups of observations, much like hand-drawn annotations done atop of a paper drawing.

5.3.4   Cluster labeling

Having found the cluster outlines $L(D_i^m)$, we now use these to position labels to explain the clusters D_i^m. As label tex, we use the names of the dimension(s) identified earlier in the top-rank sets for each cluster (see Sec. 5.2.2). For single-dimension explanation, a single label per cluster is created; for dimension-set explanation, multiple labels per cluster are used.

For label placement, we adapt the technique of Paulovich *et al.* used to create tag clouds for the similar purpose of explaining multidimensional projections [143]. This algorithm expects as input a 2D shape, for which we use our outlines $L(D_i^m)$; and a set of weighted labels, for which we use the names of the top-ranked dimensions weighted with their corresponding ranks μ_i^j, which are identical for all points in a cluster by construction, see Sec. 5.3.2. Labels are next

scaled (in font size) by their weights and placed so as to reside inside the out-line. In brief, the algorithm builds an axis-aligned uniform grid over $L(D_i^m)$, after which labels are placed, in decreasing weight order, over free grid cells as close to the centroid of $L(D_i^m)$ as possible. If a label cannot find a valid place, due to the lack of available free cells, all label sizes are decreased and the placement re-starts, until all labels are placed.

The original label placement in [143] can position labels both horizontally or vertically, much in the spirit of tag clouds. We have experimented with this place-ment, but found that it can create label sets which are hard to read due to the changing text-reading direction. For our context, we chose to place all labels for a given cluster at the same angle. This favors visually comparing the font-sizes for labels in the same cluster, which is essential for quickly telling which dimension is most important for describing that cluster. Additionally, this simpler layout lets us design the placement of dimension-value bars, explained next in Sec. 5.3.5. As an extension to [143], we allow placing labels at arbitrary angles, rather than only horizontally or vertically. The placement angle, or vector parallel with the text di-rection, is given by the direction of the major eigenvector of the covariance matrix of the sample points in $L(D_i^m)$. Figure 5.6 shows an example of automatic label-ing for the well-known *segmentation* dataset [6] projected using LAMP [95]. As visible, labels are well-centered inside their respective clusters and also oriented to take maximum advantage of the cluster shape.
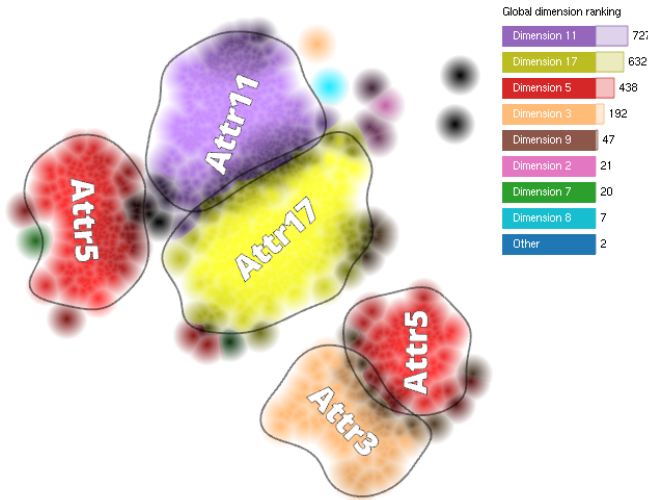
Figure 5.6: Automatic labeling for the *segmentation* dataset.

Cluster labeling has several key advantages as opposed to the basic color coding introduced in Sec. 5.2.2. First, it can accommodate significantly more dimension-names than the small number of colors ($C = 9$) present in the categorical col-ormap used for dimension color coding. This added-value increases as we move from single-dimension to multiple-dimension explanation, where more colors would

needed to cover the typical number of dimension-sets required for an explanation. Secondly, one can directly read the top-ranked explanatory dimensions, and their relative importances, in terms of labels and font sizes *on the projection* itself, rather than having to correlate colors from the projection to the color legend. This is particularly helpful for disambiguating potential confusions in matching colors from the projection to the color legend, due to the brightness encoding of confidence (see discussion at the beginning of Sec. 5.3).

5.3.5   Dimension-value explanation

Our visual explanation techniques so far partition the projection space into clusters explained by dimension *names*, indicating in which such dimensions are points in a cluster most similar. However, this explanation can create several same-hue regions in a projection – see *e.g.* the two purple regions explained by the *lines of code* dimension in Fig. 5.2c: These are point groups being similar, indeed, mainly from the perspective of the same dimension, but having different dimension *values*.

We solve this issue by showing dimension values using a color-legend bar besides the textual (label) cluster explanation discussed in Sec. 5.3.4. The legend follows the model proposed by Oliveira *et al* [134]: A 1D bar is drawn, partitioned into maximally four parts, colored using an ordinal colormap (Fig. 5.7a). Each part maps an equal-sized interval between the minimum and maximum value, for a given cluster, of the respective dimension. The length of each part shows the number of data values in that range. Three numerical labels atop the bar show the minimum, average, and maximum values of the data in the cluster. In other words, this bar represents a compact version of a four-bin histogram of the data values.

Dimension-value bars are shown on demand when the user brushes a dimension label in a cluster, so as to not needlessly clutter the visualization. Figure 5.7b shows an example of dimension-value bar for the cluster best explained by Attr17 in the *segmentation* dataset, which is also shown in Fig. 5.6. We see that the minimum value for Attr17 in the brushed cluster is 0.1, attained only for few data points (short dark brown bar segment). Almost half of the points have values varying from 25% up to 50% of the maximum of Attr17, as shown by the dark orange bar segment. The remaining (roughly half) points have values from 50% to 75% of the maximum of Attr17 (light orange bar segment).

5.4   Example Applications

To illustrate the working and value of our entire set of visual explanatory tools, we use them to address three different analysis tasks for three real-world datasets (Secs. 5.4.1-5.4.3). As projection technique P, we now use t-SNE [193] instead of LAMP, which has been used in our examples so far. This choice has two motiva-

(a) Categorical colormap for dimension-value ranges

(b) Value ranges for dimension `Attr17` over the brushed cluster
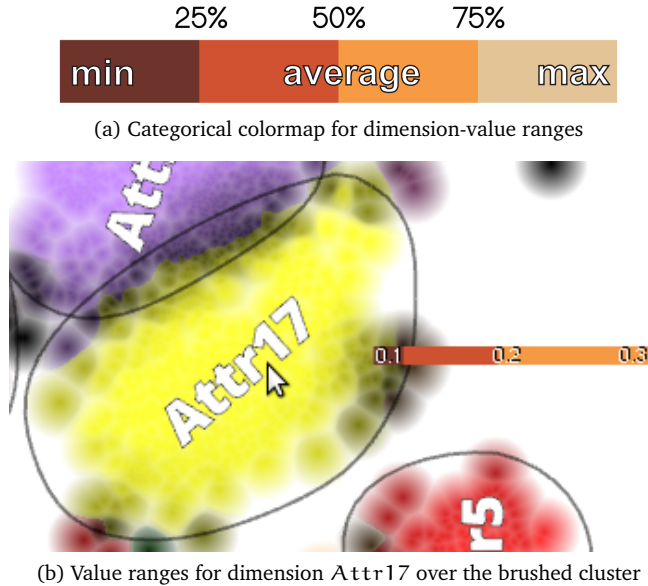
Figure 5.7: Visualizing values of dimensions over a point cluster.

tions. First, from an application viewpoint, t-SNE is arguably better than LAMP, as it favors a better separation of groups of similar observations in the projection image (in case such groups exist in the data). Secondly, from a validation viewpoint, we want to show that our explanatory techniques do not depend on the choice of a projection technique. Showing that we can handle t-SNE is a particularly interesting test. Indeed, as outlined in Sec. 5.3, t-SNE does not aim to preserve Euclidean distances (like LAMP) but neighborhoods. However, our explanatory techniques are based on the use of distance-based metrics (see Sec. 5.2.1, can be challenging). Hence, it is interesting to see whether we can use our techniques for a projection which does not explicitly aim to preserve distances.

5.4.1   Handwritten digits dataset: Finding discriminative dimensions

This dataset describes numerical digits handwritten by 44 human writers [2]. Each writer was asked to write 25 samples for each digit (class) in random order using a pressure sensitive tablet. Each digit is represented by a feature vector of 16 dimensions, defined by a sequence of 8 coordinates (x, y) captured by the tablet. Such feature vectors are typically used to build classifiers for automatic handwriting recognition. A key challenge here is to find out which features (and feature values) are best to discriminate the various digit classes. This so-called *feature selection* task is standard in the design of classification systems [104].

To help answering the above, we selected 100 random samples for each digit class, yielding a total of 1000 data points. Figure 5.8a shows the resulting projection, done using the 16-dimensional attributes only (the class attribute is not
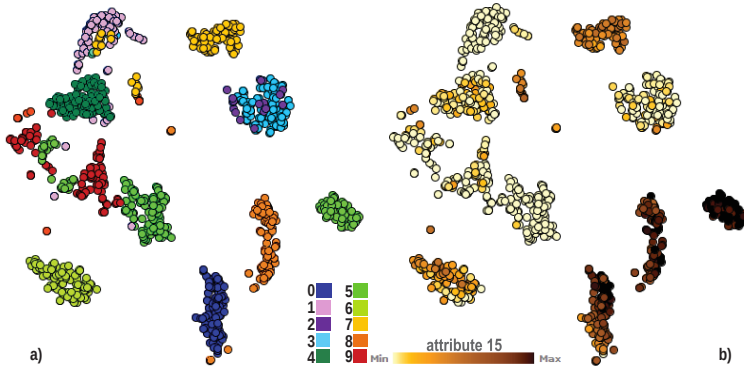
Figure 5.8: Handwritten digits dataset. Projection colored by (a) class IDs and (b) values of attribute 15. See Sec. 5.4.1.

used), and color-coded by class, using a categorical colormap. We see five of the ten classes are well separated into five visual groups: digit 0 (dark blue), digit 5 (light green), digit 6 (lime), digit 7 (light orange) and digit 8 (dark orange). The five remaining visual groups contain a mix of the remaining five classes: digit 1 (pink), digit 2 (purple), digit 3 (cyan), digit 4 (dark green), and digit 9 (red). Assuming that the projection captures well the 16-dimensional similarities of the data points, this shows that separating certain digit classes from other classes is hard.

As such, it is interesting to find out which dimensions contribute to the good (or bad) class separation. For this, we start with the single-dimension explanation (Fig. 5.9a). We see here that digits 5 and 2 are best explained by similar values of dimension 14 (yellow group). Digit 7 is explained by similar values of dimension 3 (beige group). Digit 6 is best explained by dimension 9 (red group), but this dimension also explains about half of the digit 8 samples. Dimension 13 (brown) explains a small subset of the digit 3 samples, with significant noise, shown by the low luminance. Dimension 15 (purple) is most relevant for digits 1, 4, and 9, and also the remaining samples of digit 8. Finally, digit 0 is explained by two groups defined by dimensions 10 (pink) and 8 (green).

By looking for bright-colored groups in Fig. 5.9a which match well-separated class-groups in Fig. 5.8a, we find that some classes can be very well separated by *single* attributes, *e.g.*, digit 7. To separate the other classes, we need, however, to use multiple attributes. We explore this using the dimension-sets explanation (Fig. 5.9b). We see here how groups of dimensions can, indeed, separate several classes, *e.g.* digit 5 (dimensions 9, 1); and digit 1 (dimensions 15, 13, 6). Interestingly, we also see that dimension 15 appears with high ranks in many far-apart groups corresponding to different digits. This suggests that this dimension is not really useful for a classification task.

Figure 5.9b also shows another visual design for the dimension-value legends introduced in Sec. 5.3.5. In contrast to the on-demand display of these legends

illustrated earlier, we use now a design where the legends are displayed for all explanatory dimensions for a set of selected clusters (three in the figure). This allows comparing the distribution of values of different dimensions across the same and/or different clusters. For this visual design to work, dimension-labels are oriented horizontally and ordered top-to-bottom in decreasing rank order per group. As such, reading the explanation of a given group is simple: Top labels give the most important dimension explaining that group, and the distribution of values for a dimension is given by the bar displayed right next to that dimension's label.

Validation: To validate our findings, and get more insight on dimension 15, we color map its on the projection (Fig. 5.8b) and compare the emerging patterns with our explanatory maps. We see a value gradient going roughly from low values (top-left) to high values (bottom-right). The direction of this gradient matches quite well the spread of the three purple groups in the single-dimension explanation in Fig 5.9a. By adding dimension value bars atop of these groups, we verify indeed that dimension 15 increases from top-left to bottom-right, and has very different value-ranges for the three purple groups. This insight is refined by the value bars shown for the finer-grained dimension-set map (Fig. 5.9b).
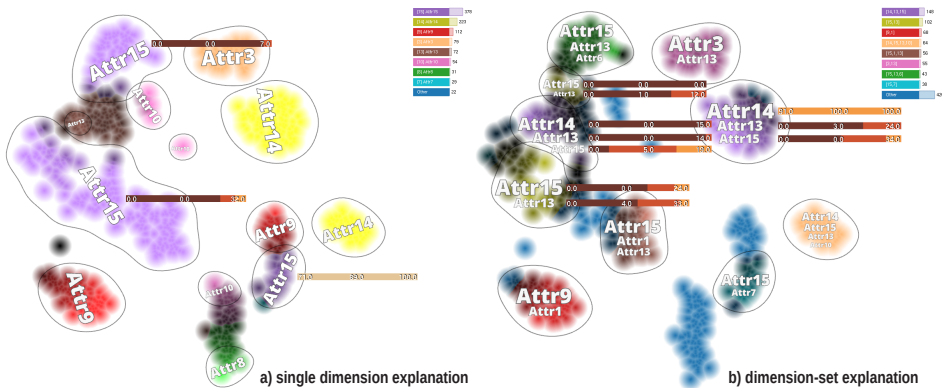
Figure 5.9: Handwritten digits projection explanation by both single dimensions (a) and dimension-sets (b).

5.4.2 Concrete strength dataset: Finding predictive variables

This dataset describes how values of eight ingredients influence the strength of concrete, a key material in civil engineering [210]. It contains 1030 samples that measure how concrete strength is affected by the mix of eight ingredients: *cement*, blast furnace slag (*BFSlag*), *fly ash*, *water*, *superplasticizer*, coarse aggregate (*Caggregate*), fine aggregate (*Faggregate*), and *age*. Figure 5.10a shows the projection of the data using only the eight ingredient attributes and colored by concrete strength. We see here several well-defined groups, and a concentration of high

concrete-strength values in the lower right. A typical question for this kind of data is to find out how the eight input variables (and their ranges) correlate with desirable values of the output variable (high concrete strength). This is a typical task in machine learning of *predicting* the value of a so-called dependent variable based on the measured values of independent variables [47].
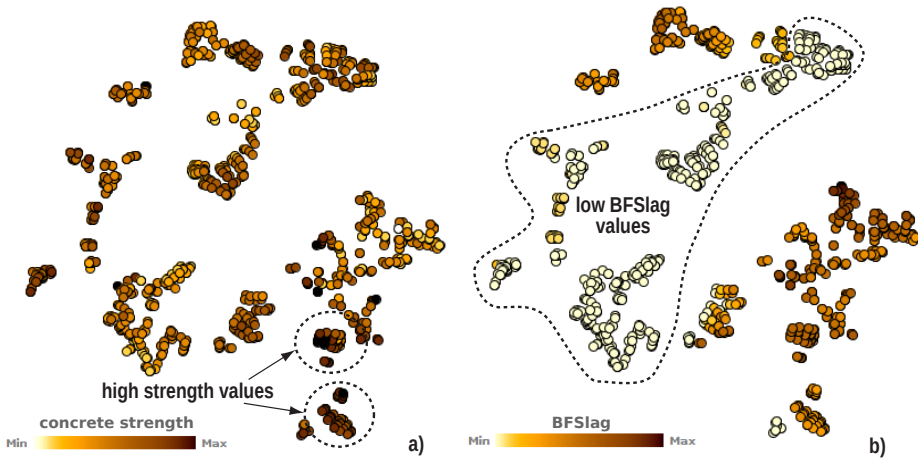
Figure 5.10: Concrete dataset colored by concrete *strength* (a) and *BFslag* (b) attributes.

Using the single-dimension explanation (Fig. 5.11a), we see two large groups, defined by dimensions *fly ash* (yellow) and *BFSlag* (purple), respectively. *BFSlag* also explains two smaller groups upper-right in the projection. Adding the dimension value bars to the three purple groups shows an increasing *BFSlag* trend top-to-bottom. Two remaining well-defined groups are explained by similar values of *age* (beige) and *cement* (red). Finally, we see a dark confusion area between the *cement* and *BFSlag* top groups, indicating points which are not clearly differentiated from their neighbors by any single dimension.

To refine these insights, we use the dimension-set explanation (Fig. 5.11b). As also seen for the example in Fig. 5.9, finer grained and better explained groups appear now. For example, the large *fly ash* group is now split into four smaller groups, which add the dimensions *cement* and *BFSlag* to the explanation. Interestingly, a large part of the bottom two groups (Fig. 5.11b, dotted circles) matches very well high-strength value areas (Fig. 5.10a, dotted circles). Exploring these groups with the dimension value bars (not shown here to reduce visual clutter) can, thus, tell us which specific *BFSlag* and *fly ash* value ranges correspond to high concrete strength.

Validation: To validate our explanatory maps, we proceed as in the previous case (Sec. 5.4.1): We color map an attribute used for the projection (*BFSlag*, which was shown to have interesting correlations with concrete strength), leading to Fig. 5.10b, and compare patterns in this figure with our maps. We easily see that

a) single dimension explanation   b) dimension-set explanation
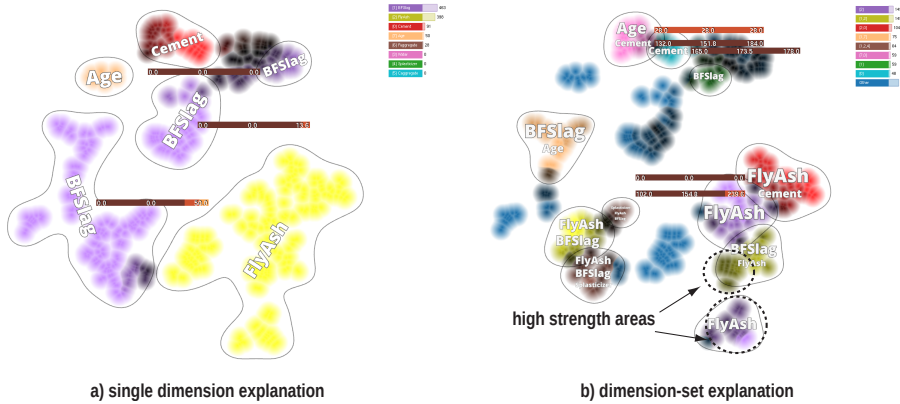
Figure 5.11: Concrete dataset projection explanation by both single dimensions and dimension-sets.

the bright band of points having very low *BFSlag* values (Fig. 5.10b, dotted area) correlates well with the extent of the three purple *BFSlag* clusters in Fig. 5.11a. Showing dimension value bars atop of these clusters indicates that they have very low *BFSlag* values, which again correlates with the color map in Fig. 5.10b. Apart from the above, Fig. 5.10b also tells us that the lower-right groups shown earlier in Fig. 5.11b, which as we have seen contain high concrete strength points, are characterized by high *BFSlag* values.

5.4.3   Forest fires: Explaining groups of observations

This dataset contains measurements in the Portuguese Montesinho park, collected daily from January 2000 to December 2003 to predict and prevent forest fires [32]. Every time a forest fire occurred (517 occurrences in total), ten features were registered. These describe the fire's spatial x and y locations in a 9-by-9 cell grid, wind speed, temperature, and the six attributes of the Canadian system [181] for rating fire danger: fine moisture code (FFMC), duff moisture code (DMC), drought code (DC), initial spread index (ISI), buildup index (BUI) and fire weather index (FWI).

We start exploring this dataset by projecting it (Fig. 5.12a). A first salient observation is that the projection is clearly split into two groups A and B. As such, we are interested to find out what 'binds' the points to form these groups. The classical option is to successively color code the projection by all attributes to find out if an attribute causes the group separation. Doing this, for instance, for *temperature* does not show a clear difference between the two groups (Fig. 5.12a). We could repeat this process for all other nine attributes, but we prefer a less tedious solution. For this, we first try the single-dimension explanation (Fig. 5.13a). Here, we notice that the placement of most points is mainly determined by the *rain* di-
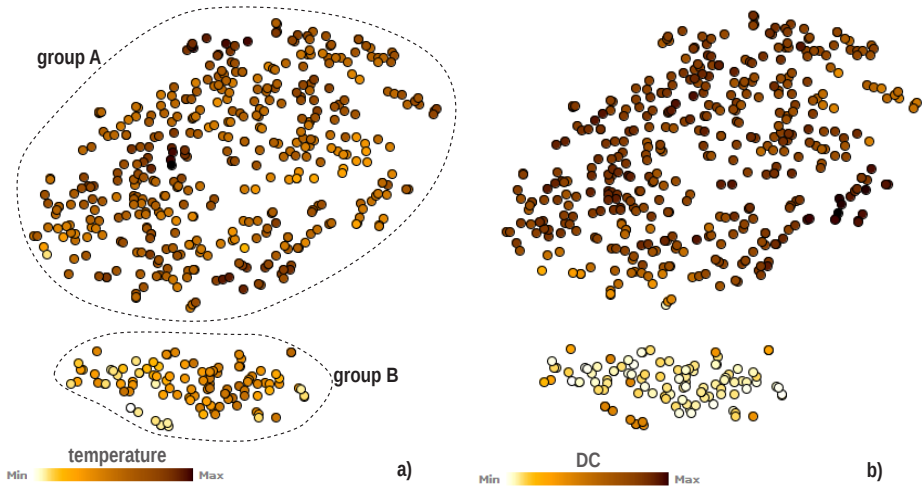
Figure 5.12: Forest fires dataset projection colored by (a) *temperature* and (b) value of the *DC* attribute. See Sec. 5.4.3.

mension – except a small well-defined yellow group that is explained by the *DC* (drought code) dimension. Showing the rain values in the two purple groups by using dimension value bars tells that these are very similar. Hence, rain does not explain the separation of groups A and B, but the separation of the yellow group from the rest.
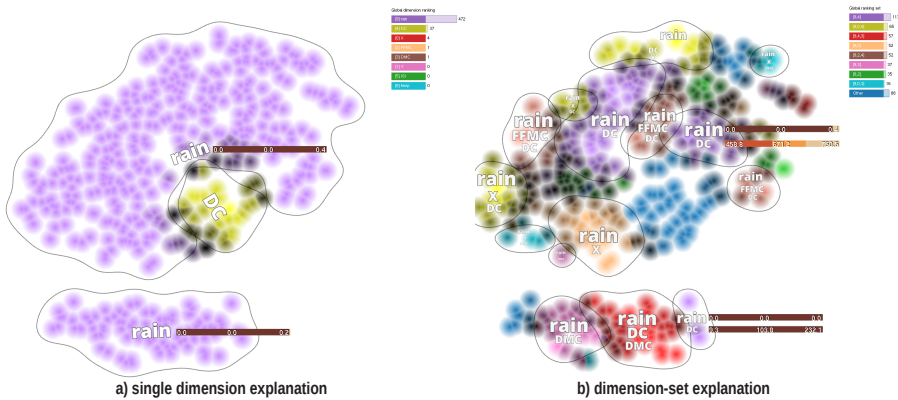
Figure 5.13: Forest fires projection explanation by both single dimensions and dimension-sets. See Sec. 5.4.3.

We next refine the exploration using the dimension-set map (Fig. 5.13b). As expected from Fig. 5.13a, we see that *rain* is the key explanatory attribute for most emerging groups. We also see how both groups A and B are now split and

explained locally in terms of several other dimensions. For instance, the pair *(rain, DC)* determines the appearance of three purple groups. Adding dimension value bars to these shows that two of these (located in the top group A) are explained by high DC values, and the third one (in the bottom group B) is explained by low DC values. Upon having seen this, we realize that, indeed, the *rain* and *DC* (drought) attributes are expected to appear together in explaining similar points, as they are inversely correlated.

To further inspect why the groups A and B are separated, we select them and use the dissimilarity ranking (Sec. 5.2.1) to explain the separation. The result (Fig. 5.14) is to be interpreted as follows: a same-color area R in group A tells why points in R are far away from the entire group B; and a same-color area in group B tells why these points are far away from the entire group A. With this rule in mind, the fact that both groups A and B are mainly covered by dimension *DC* (purple) means that they are chiefly separated because of different *values* of this dimension. Bringing up dimension value bars on the two purple areas shows that this is the case. Additionally, we see a red DMC group split from the top group A. This means that the respective points are placed far away from the bottom group B because mainly of the DMC dimension.
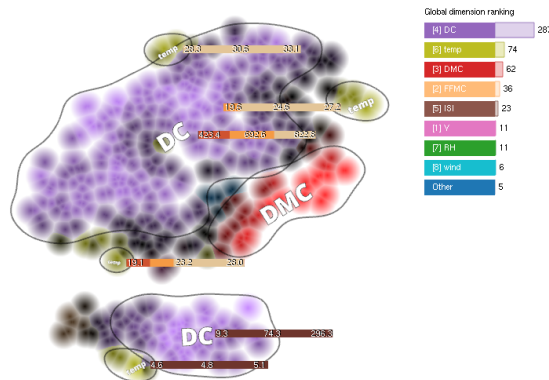
Figure 5.14: Explaining the separation of two groups in the Forest fires projection.

Validation: Checking whether our explanation for the separation of the two groups holds is simple – we color the projection by the DC attribute (Fig. 5.12b). We now easily see that the top group has high DC values and the bottom one low DC values. Hence, DC is indeed a chief explanation for the groups' separation.

5.5   Discussion

We next discuss the main aspects of our visual explanatory method for projections.

Way of use: Our explanatory method can be used to support various tasks involving the analysis of multidimensional data, as shown in the application examples in Sec. 5.4. These range from a general high-level characterization of apparent point clusters in the projection to finding more fine-grained correlations between observation classes and dimension values. In turn, these support high-level tasks such as feature selection for the design of classifier and predictor systems. For finer-grained insight, our methods should be combined with additional exploration tools such as classical brushing and color mapping data by values of a user-selected dimension.

Advantages: Our method is easy to understand, easy to use, computationally efficient (runs in real-time for datasets up to 10K points on a typical PC for a C++ CPU single-threaded implementation), and generic (can use any projection and/or dataset with quantitative dimensions). Importantly, the projection technique can be used as a black box, *i.e.*, without making any assumption about its internals or characteristics, *i.e.*, whether the projection is linear or not or global or local. Martins *et al.* have shown that such characteristics of explanatory techniques for projections are very important for their ease of use [118]. The partition of the 2D projection space into same-explanation regions is automatic. Next, this partition can be viewed *implicitly*, *i.e.*, as a colored image consisting of several same-color zones; or *explicitly*, *i.e.*, as a point set being partitioned into disjoint clusters. The implicit view is useful when one requires a high-level overview of the entire projection 'landscape', such as in presentations. The explicit view is useful when one needs to reason in more depth, and about specific groups of points in the projection.

Limitations: Color-coding explanations are inherently limited to the maximum number of colors that a categorical colormap can reasonably have. This can often be less than the number of regions we can detect in a projection. Separately, the proposed metrics for ranking dimensions (Sec. 5.2.1) are sensitive to local and global variance of the data, and also are not well suited to work very high-dimensional datasets (hundreds of dimensions or more). Better explanation metrics can be envisaged for 2D neighborhoods, *e.g.* based on feature scoring and discrimination techniques [152]. On the positive side, however, we should note that our visual explanatory techniques can easily accommodate using any such feature scoring metrics instead of our proposed Euclidean and variance ranking, with no changes to their implementation.

A separate high-level limitation relates to the power of our visual explanations in supporting complex end-to-end applications such as classifier or predictor system design. As the examples in Sec. 5.4 show, our explanations can form a *part* of the data sensemaking required to get insights for the construction of such complex systems, such as finding correlations, and dimensions and their values that explain observation groups. However, many other elements are required to construct a

good classifier or predictor, such as the selection of suitable machine learning algorithms; the selection of suitable training datasets (for supervised methods); and the fine-tuning of the parameters of such systems for obtaining optimal performance. As such, albeit we argue that our techniques can help understanding multidimensional data, we should recognize their limitations and limited scope too.

5.6 Conclusion

We have presented a set of simple and automatic techniques that visually explain 2D scatterplots created by multidimensional projections. Explanations take the form of partitioning the projection into compact and well-defined areas which are next annotated by color coding, outlines, labels, and dimension value bars. All these elements support answering the questions of which attributes and/or attribute values make points be similar, or be separated, in a projection. Answering such questions helps higher-level tasks in multidimensional data understanding in the scope of machine learning related applications. We demonstrated our methods on several real-world multidimensional datasets projected by three types of dimensionality-reduction techniques.

Our explanatory tools can be extended in several directions, as follows.

First, extending the ranking methods used here to handle datasets of hundreds of dimensions, *e.g.* by combining different types of feature ranking metrics, would make our visual explanations more useful for more complex datasets. A challenge here is that the current ranking metrics we use essentially look, for each neighborhood, for the axis (or axes) in n dimensions along which the neighborhood has a small projection – this is the essence of dimensions having a high rank. As the number of dimensions of a dataset grows, finding such axes is increasingly unlikely. Moreover, other significant data patterns can exist over a neighborhood, *e.g.*, the fact that its data lives on a low-dimensional manifold, which is not necessarily oriented orthogonal to some axes. We imagine that specific detectors for such local data patterns can be designed, followed by specific visual encodings for their presence. This would make the specificity of our visual explanations much higher than it currently is.

Secondly, we believe that the dimension and dimension-set explanations presented here could be generalized to produce a multiscale (hierarchical) explanation of large projections in terms of a set of nested regions, ranging from local and precise explanations to more global and simple ones. This would allow exploring different areas of a projection at different levels of detail, thereby combining summarization with precision in the same view. Visualization-wise, the envisaged look and feel if such multiscale explanatory maps could resemble that of cushion treemaps [196], given the cushion-like luminance profile formed by our color-and-brightness encoding, and the perceived spatial nesting of specific-explanation zones within more-general-explanation zones. This analogy suggests that our vi-

sual design would scale well to large projections consisting of tens of thousands of observations.

Finally, extending our visual explanatory techniques to time-dependent multi-dimensional datasets is an interesting (and useful to solve) challenge that would broaden the applicability scope of our proposal. This last challenge forms the goal of our next chapter.

Explanation of Time-Dependent Projections

<div style="text-align:right">6</div>

In Chapter 5, we have shown how projection scatterplots can be enhanced by various visual metaphors so as to enable users to explain the structure of the projection, in terms of groups of observations, by means of dimensions and dimension values of the original high-dimensional dataset. The respective metaphors have been demonstrated by the visual analysis of several real-world datasets.

As noted in the conclusions of the same chapter, several extensions are possible to the proposed visual metaphors. In this chapter, we focus on one of them – the visual explanation of time-dependent, or dynamic, multidimensional datasets. Similar to the static (time-independent) datasets discussed so far in this thesis, dynamic datasets consist of observations having a (large) number of dimensions. However, in contrast to static datasets, the values of the dimensions for dynamic datasets change in time for the observations they describe. Such datasets can be visually depicted by means of multidimensional projections, just as for static datasets. However, two new challenges occur here:

- How to handle the fact that the dimensions of the observations change in time when generating multidimensional projections?

- How to explain patterns present in the projection, such as groups of observations and outliers, from the perspective of the (changing) dimensions?

In this chapter, we aim to answer the above two questions, thereby extending our visual explanatory means from the static to the time-dependent domain. To do so, we first need to choose an application domain which delivers us rich multidimensional dynamic datasets. We choose here for program comprehension during software evolution, a branch of software maintenance, and discuss the types of datasets related to our goal that this domain delivers, as well as the concrete questions that this application domain spawns regarding our dynamic multidimensional datasets. These issues are discussed in Sec. 6.1. Next, we proceed by discussing related work from the perspective of the chosen application domain (Sec. 6.2). Section 6.3 discusses the process we use to acquire dynamic multidimensional datasets from a corpus of software undergoing evolution, obtained from software repositories. Section 6.4 contains the core of our technical contributions which aim to answer the top-level questions listed earlier in this section. We exemplify the working of our proposed techniques by the visual analysis of the evolution of two real-world software repositories in Section 6.5. Section 6.6 discusses various technical aspects of our proposed techniques. Finally, Section 6.7 concludes this chapter, which is also the last chapter introducing technical contributions in this thesis.

6.1   Program Understanding in Software Evolution

A key issue in software engineering is to understand how a project is organized during its development lifecycle. To this end, two main approaches can be taken. First, one can analyze the changes of the so-called *explicit* structure of the project, captured by its physical or logical hierarchy and dependencies [20, 184, 83]. Alternatively, one can mine the repository to find so-called *implicit* structure, *i.e.*, aspects of the recorded data which create groups of highly-related entities. Such aspects can be inferred by finding groups of highly-similar software entities from the perspective of their quality metrics [111], source code (*i.e.*, clones) [156], co-change [3], and lexical term frequencies [106]. Analyzing changes in the explicit project structure is the by far more common approach in program comprehension, and is motivated by the relative ease of extracting and presenting change data along well-known project subdivision axes, such as folders and files (physical hierarchy); namespaces, classes, and methods (logical hierarchy); and call, dataflow, include, and inheritance relations (dependencies). Many methods have been designed to extract and present information along such axes in the scope of what is commonly known under the name *software visualization* [44, 122]. In contrast, detecting implicit structure can uncover previously unknown patterns and changes in the software under study, since the search for such patterns is not constrained to follow a given organization of the project. Additionally, presenting data obtained from this perspective can offer additional insights in the evolution of the software. However, detecting (or mining) and presenting such implicit aspects is harder, since the search space is larger.

One well-known way to characterize software and its evolution is to extract so-called software quality metrics and study their values, distribution, and changes [111]. Conceptually speaking, this process is identical to the well-known feature extracting used to characterize other large collections of complex entities, such as text or images, which is well known in machine learning and pattern recognition [46]. The essence of this process is as follows: Given a collection of such complex entities, one aims to find patterns such as groups of similar observations, outliers, and changes into the above, as data evolves in time. However, one usually does not know which aspects are key to capturing such patterns. As such, a number of *features* are extracted for each item, in the hope that values (and changes) of these will capture the essence of the underlying processes. Such features are typically called *software metrics* in program comprehension. Since a key goal in software engineering is the increase of quality of produced software, most such metrics relate to the quality of software, and as such are called software quality metrics [111]. Once the data collection has been reduced to a multidimensional observations, its study follows the traditional pattern of analyzing multidimensional datasets.

If we focus on aspects implied by software quality metrics and their change, a challenging point is the large number and variability of such metrics. Typical software static analysis tools deliver tens of such metrics, each capturing a different facet of the software [163]. Understanding how entities in a software system re-

late to each other, according to their metric values, and how such relations change in time, imply understanding how tens of measurements, recorded on hundreds of items, change over hundreds of revisions in time. Reducing the amount of data to analyze can be done, but is risky. For instance, one can analyze the evolution of a single (or a few) metrics over all entities. This process is similar to what one would call feature selection in machine learning [74]. If done arbitrarily, *e.g.* by manually selecting a few metrics to analyze [199], this can easily miss important underlying aspects relating the entities, which are captured by the metrics not considered in the analysis. Conversely, analyzing the evolution of all metrics over a few entities offers only a coarse partial view of the existing aspects and their change [189]. To understand the full data space, we essentially have to understand the evolution of a multidimensional dataset D of n metrics, measured on m entities, over T time moments, for large values of n, m, and T. In turn, to do the above we need ways to capture and depict metric-implied patterns formed by groups of entities, and ways to track pattern changes over time.

We approach this problem by proposing Metric Evolution Maps (MEMs), a novel approach for the visual exploration of multidimensional time-dependent datasets. We use software quality metrics, mined on all entities (classes) of all revisions of a repository to detect groups of highly-similar entities in revision, by using multidimensional projection techniques. Next, we explain these groups in terms of their shared metrics and metric-value properties. Finally, we explain change patterns at entity and group level in terms of the underlying metric changes. Overall, the presented techniques aim to answer two types of questions:

Q1: How do entities group in a given revision? Which are the main groups? Which metrics determine these groups?

Q2: How do groups change in time? How do entities migrate between groups, and due to which metric changes?

To address the above, we propose two types of contributions. First, we enhance multidimensional projections with visual explanatory tools so as to support Q1. For this, we use the visual explanatory techniques proposed in Chapter 5, suitably adapted to handle projections that come from time-dependent datasets. Secondly, we enhance these techniques with additional visual explanatory tools to support questions regarding evolution (Q2).

The remaining of this chapter is organized as follows. Section 6.2 overviews related work on visualization of the evolution of software metrics and related artifacts. Section 6.4 describes our approach. Section 6.5 shows how MEMs can be used to discover several aspects, and their changes, in two real-world software repositories. Section 6.6 discusses our results. Section 6.7 concludes the chapter.

6.2   Related Work

Many techniques have been proposed to visualize similarity (and changes) in the structure of code over various types of entities such as code lines [201], syntactic blocks [184], hierarchies [20, 83], and code clones [75]. Such techniques typically consider a single or a very few attributes to capture and detect change. As such, many aspects that induce similarity (or changes) are not captured. In the same time, these techniques organize the data to be presented along the lines of the explicit structure of the software system. For example, the visualization tool presented in [201] organizes data mined from a software repository along lines of code, files, and folders. Other software visualization tools organize data along hierarchical physical structure [20, 83]. This structure consists of a so-called compound graph [44] which in turn consists of containment edges (typically folders, files, and methods) and dependency edges (typically function calls, inheritance relations, type usage, and dataflows).

As already mentioned in Sec. 6.1, a major advantage of this organization of the data is that it is *familiar* to the developer interested to study the software. Indeed, one will arguably easily recognize the main folders or classes in a software system one works on. However, a problem of this organization is that aspects which do not follow this structure can be hard to find. For example, consider the task of finding classes which have a similar evolution in time in a software project. If classes are depicted along a physical hierarchy, then one will be able to easily compare classes located in the same folder or folders close to each other in the physical hierarchy; but it will be harder to spot correlated changes in classes located far away from each other in the project structure. These aspects have been recognized in program understanding, and various techniques have been designed to search for *similarity* relating any set of entities in a software project. One of the best known class of techniques in this area is detecting duplicates, or software clones [156]. These are code fragments which have a high lexical or structural similarity, and can occur in any parts of a software project. Code clones can be visualized by various techniques that essentially depict a compound graph such as hierarchical edge bundles [200], polymetric views [49], Hasse diagrams [94], and adjacency matrices [98]. From our perspective of being able to easily explore similarity relations, we can argue that such techniques go only half-way: While clone relations are, indeed, mined between any entities of the considered dataset, their visual presentation still follows the explicit software structure. As such, the visual distance between entities does not usually reflect their similarity. As we have seen in Chapters 3 and 4, making the visual distance reflect data similarity is a powerful cue to exploring similarity relations.

At the other end of the spectrum of analyzing software, static analysis and repository mining can be used to extract tens of metrics such as code size, complexity, cohesion, coupling, number and type of bugs, and identity of developers changing it [111, 122, 199]. While such approaches provide very rich characterizations of the underlying software entities, depicting patterns (of similar and/or

related entities) and their changes is hard, due to the high dimensionality of the data. Several techniques attempt to depict such high-dimensional data, with various degrees of success, as follows. Space-filling approaches such as table lenses [154], evolution lines [201], and evolution matrices [110] use a 2D Cartesian layout populated with sparkline-like encodings [190] to show the variation of metrics *vs* entities *vs* time. Variations include 3D Cartesian layouts and UML layouts using bar charts [109, 62]. However, such approaches typically cannot show the *entire* data space in case of hundreds of entities, tens of metrics, and hundreds of change moments. Moreover, finding groups of similar entities, or understanding how entities change in time with respect to each other and with respect to the underlying metrics, is hard, due to the usage of a visual layout which does not essentially reflect similarity, but software structure.

The challenge of showing similarity, as given by many software metrics, has been recognized in program understanding and software visualization, and several specific approaches have been proposed to address it. In this class, the most prominent ones use a map metaphor to place entities so as to reflect their similarity in terms of metric values. Several such approaches employ Self-Organizing Maps (SOM) to create 2D visual representations of multidimensional data [145, 116, 153]. The key idea is to create a neural network, and dispose it in a 2D grid. After the network training, each multidimensional input data is disposed in one cell of the grid, according to the most similar neuron. Similarity can be next visualized by analyzing clusters of 2D elements. However the user must define the number of nodes (or cells) in the map to give the desired level of granularity, and the underlying number of clusters in a dataset might not be known beforehand. Also, SOMs are not always perceived as being very intuitive. More details on SOMs are given in Section 2.4.1. Other ways to construct software maps create a force-directed layout of a graph whose nodes are software entities and edges capture relations of interest. If edges are weighted by similarity, the resulting layout will naturally pull similar entities close to each other. Based on this layout, a density map encoding one metric of interest can be next computed and displayed to show so-called 'hot spots', *i.e.* areas containing many related entities that share high values of the metric of interest [195, 186].

Other approaches use multidimensional projections (MPs) to map multidimensional data into the visual space: For a set of entities $E = \{e_i\} \subset \mathbb{R}^n$ (having n real-valued metrics each), an MP creates a set of typically 2D points $P = \{q_i\} \subset \mathbb{R}^2$ so that the pairwise distances $\|q_i - q_j\|$ are as close as possible to $\|e_i - e_j\|$. The resulting 2D scatterplot-like image P can then be used to find groups of similar entities as well as outliers in E. Many MP techniques exist that score highly in distance preservation for a *static* dataset E, *e.g.*, ISOMAP [188], LAMP [95], LSP [141], and t-SNE [193], to mention just a few. More details on MP techniques are given in Sec. 2.4 and throughout the previous chapters of this thesis.

When compared to other high-dimensional visualization techniques such as table lenses, evolution lines, evolution matrices, parallel coordinates [90], and scatterplot matrices [78], MPs are considerably more scalable in number of entities m

and dimensions n, and easier to use to find groups of related entities. However, they have two challenges:

Group explanation: MPs do not show by default which dimensions are responsible for the appearance of groups. Our techniques introduced in Chapter 5 aim to address this goal. However, these techniques were developed for the analysis of a static projection. As such, it is not clear how they fare, or need to be extended, to handle dynamic datasets.

Evolution: The vast majority of MPs do not handle dynamic datasets E_t. We know only two exceptions, as follows.

Kuhn *et al.* [106] use multidimensional scaling (MDS) [177] to show the evolution of lexical similarity of source-code entities, computed based on term (identifier) frequencies processed by Latent Semantic Indexing (LSI) to factor out synonymy and polysemy. To make the emerging projections P_t consistent over time, they propose two variants – offline MDS, *i.e.*, projecting the union of revisions $\bigcup_{1 \leqslant t \leqslant T} E_t$; and online MDS, *i.e.*, constructing P_{t+1} by projecting E_{t+1} using P_t as an initialization of MDS. Additionally, they note that term frequencies typically change slowly in time over the same software corpus – a situation that clearly does not hold for software metrics in our case. Both above MDS variants have issues: Offline MDS will generate high projection errors for a large number of time steps T, as it tries to preserve distances between points in *any* two revisions E_{t1} and E_{t2}. Online MDS that starts with a previous dataset-projection (as opposed to random initialization) is strongly biased and can easily converge in a local minimum. Both above issues are discussed for the well-known t-SNE MP technique [193] by Rauber *et al.* [151], and apply to MDS as well.

The recent dynamic t-SNE (dt-SNE) technique in [151] is, to our knowledge, the only MP for time-dependent datasets that offers explicit and verified guarantees in terms of spatial and temporal coherence. Trade-off between preservation of distances in the same projection *vs* preservation of distances across projections which are close in time is controlled by a user parameter. However, dt-SNE has not yet been used for software evolution exploration or, for that matter, for the visual exploration of complex real-world datasets. Also, change patterns are depicted by animation, which makes spotting such patterns quite hard for more than a few time frames P_t.

The following two sections outline our proposal to construct a visual exploration system for the similarity-based study of the evolution of software entities in a software repository. The presentation is divided into two steps, along the typical construction of a software visualization solutions: First, we describe how we gather software quality metrics from a given software repository, thereby creating our dynamic multidimensional dataset to be explored next (Sec. 6.3). Next, we describe the extensions to the visual explanation techniques in Chapter 5 that we propose to visualize our dynamic multidimensional dataset (Sec. 6.4).

6.3 Software quality metrics extraction

Software quality metrics aim to measure different characteristics of software entities (e.g. classes). Several metrics exist, providing a quantitative measure of degree of a characteristic to a software entity. These measurements can be used by organizations to control or draw conclusions about a project and its development process. For example, some metrics can be used to evaluate the development of new products, by inspecting desirable characteristics of successful products, and next aim to achieve similar measurements on a new product. Other metrics, like source code size and complexity, can be used to estimate software development costs [112] or estimate how error-prone it can be.

Our goal is to understand the implicit structure of software entities provided by similar metrics values. These are computed from source code, can be quickly generated, and there are several automatic tools for that. Our approach can, however, be employed on any quantitative measurements, including dynamic ones (which are based on software execution, and thus are more difficult to extract). Specifically, we focus on the analysis of software metrics of three kinds: complexity, volume and object oriented metrics, as follows.

Complexity: These metrics are related to the measurement of the complexity of information flow and organization of code. Examples of such metrics are *cyclomatic complexity* and *depth of conditional nesting*. The cyclomatic complexity measures the number of independent paths on the execution of code, providing an upper bound of the number of tests to be executed in an entity to fully cover all its execution paths. The depth of conditional nesting measures the depth of nested `if-then-else` statements. Large values for this metric might indicate potentially error-prone software, since it is easier to introduce flow-related bugs on deeply nested conditionals.

Volume: These metrics relate to the measurements of software size. Examples are KLOC (thousands of lines of code), number of files, number of statements, number of blank lines, and number of line comments. Volume metrics are typically measured per unit of system structure, such as methods, classes, or files. High values of these metrics indicate large entities, which are typically harder to understand and maintain, and indicate potential needs for refactoring.

Object oriented: These metrics aim to capture the maintainability and understandability of object-oriented software. They extract information at the level of classes, methods, or packages (groups of related classes). Examples are Lack of Cohesion in methods (LCOM), Coupling Between Object Classes (CBO) and Number of Children (NOC). High cohesion in object-oriented programming is important, since it helps promoting encapsulation. The LCOM metric indicates how much methods are different in a class by means of its shared attributes, thus measures the opposite of cohesion. Coupling measures the number of elements that

relate two software entities such as classes; such elements can be function calls, direct reads and writes to data members, or usage of locally defined data types. Ideally, coupling should be low, as it promotes encapsulation and decouples the maintenance and understanding of different classes. High coupling indicates that changes in one class will affect many other classes, thus, demanding more effort on code maintenance.

To characterize software, we extract a large number of software quality metrics from all above-mentioned types from a given software project. To characterize evolution, we apply the above process to consecutive versions, also called revisions, of a software project. Since versioned software typically comes in software repositories, such as CVS, Subversion, or Git, we use such repositories as the primary source of information to collect our dynamic multidimensional metric datasets. The process of creating such datasets from a given repository is described next. Specifically, Sec. 6.3.1 describes the choice of repository type and handling of its multiple revisions. Section 6.3.2 describes the process of extracting metrics from a single revision.

6.3.1   Extracting metrics from software repositories

The first choice to be made here is to select the type of source control versioning system, or type of software repository, as it is more commonly known in the software engineering practice. As mentioned in the previous section, several types of software repositories exist and are widely used in practice – the best known being CVS, Subversion, Git, CM/Synergy, and Microsoft's Team Foundation Server (TFS). From the above, we excluded first CM/Synergy and TFS, since these are commercial systems, so the vast majority of software projects managed by them are closed-source, thus not open to us for analysis. From the remaining open-source repositories, we excluded CVS, as being relatively outdated and less frequently used nowadays. Comparing the remaining alternatives (Subversion and Git), we note that Git [24] has several advantages: It is fast, since the whole repository can be downloaded and explored offline, avoiding unnecessary queries to a network server to collect information. Git stores file changes using an efficient file system, where each revision (commit) is seen as a snapshot of the current project state. Files that do not change between revisions are not stored redundantly, which saves space and allows efficient search throughout the revision history. Git also provides a well documented library[1] to explore the revision history.

A second choice to make is the type of software, in terms of programming language, to explore next. We focused on exploring projects developed in C++ and Java. These are very popular programming languages with many open source projects available for download in hosting services such as *github* and *bitbucket*. Furthermore, these are object-oriented languages, which offer us more metrics to analyze than projects written in procedural languages such as C.

---

1 libgit2 – http://libgit2.github.com

Once a project is selected for exploration, we create a directory hierarchy on disk to represent the revision history. This is a two-level tree, where siblings on the second (leaf) level represent the different revisions to analyze. Each revision directory contains all files of the respective revision. Files that do not change with respect to an earlier downloaded revision are represented here as symbolic links. This way, we achieve two desiderates: (a) we minimize the storage space on disk for large repositories; and (b) we offer, per revision, a directory that contains all the files (either stored physically or as links) in that revision. This way, we can directly use existing software metrics extractor tools on every single revision of the repository, even if such tools are not designed to handle versioned data.

To extract metrics from a given Git repository, our process requires specifying the repository name (URL), a set of specific file names to analyze from the repository (by default, all files written in C++ and Java are selected), a start revision r_S, an end revision r_E, and a number S of time samples in $[r_s, r_e]$ to analyze. Next, we extract metrics from S revisions uniformly distributed, time-wise, in the interval $[r_s, r_e]$. This allows specifying a trade-off between precision and processing time when analyzing large repositories having thousands of revisions and thousands of files per revision. The extracted metric values are saved next as a set of S comma-separated-values (CSV) text tables, one table per analyzed revision. Table rows describe the entities that have been analyzed, such as classes, files, or packages, depending on the capabilities of the selected analyzer tool (discussed next in Sec. 6.3.2). Table columns describe all metrics that the analyzer extracted, one column per metric. All tables share the same number of columns. In other words, we extract the same set of metrics from all analyzed revisions, their exact number and identity being given by the capabilities of the selected analyzer. Tables can however have different rows, depending on the presence of specific entities in different revisions, due to changes such as additions and deletions of code. The set-up of our repository-wide analysis is very similar to other tools that extract metrics from software repositories, such as ClonEvol [76] or SolidTA [154]. The main difference is that we do not use a relational database to store the extracted values. This decision is motivated by implementation simplicity; since we do not need to execute complex queries on the extracted metrics; and since we only store quantitative attributes (metric values) and text attributes (software entity names), whereas the other aforementioned tools need to store a much more complex set of artifacts, such as categorical and relational attributes.

6.3.2 Tools for metric extraction

Given the analysis set-up described in Sec. 6.3.1, our next choice to be made regards the tool(s) to use to extract software metrics from a single revision.

Software metric tools have a long history and have been extensively used in both research and industrial practice [111, 122, 117, 61]. However, surprisingly enough for such an established field, it is still far from evident how to choose a

'good' metric extractor, given a number of requirements it should satisfy. In our context, these requirements are as follows: (i) fully automatic execution, *i.e.*, ability to execute in batch mode, since we need next to extract metrics from hundreds of revisions of a repository; (ii) fast metric extraction, since we need to process potentially thousands of revisions having thousands of files each; (iii) ease of use and good documentation; (iv) extraction of a large number of quality metrics, since we aim to visually analyze high-dimensional datasets; (v) ability to handle code that does not build, since many repositories contain such code, *e.g.* they miss libraries or build rules; (vi) ability to extract metrics at a fine-grained level of detail, *e.g.* per class, file, or method, so that we have enough observation points in our multidimensional dataset to be able to reason about intra-project similarities; (vii) ability to deliver the metrics in a simple-to-parse file format, which we need to easily gather the metrics for further use in our pipeline.

Given the above, we approached the process of selecting a suitable metric extractor by practical testing. We tested 7 popular tools for extracting metrics, based on their prominence in the software metrics literature and/or their presence in various forums on the internet dedicated to metric extraction. We next compared these tools based on the above-mentioned set of requirements. As comparison material, we used several open-source projects of sizes varying from a few thousand lines of code to over 100 KLOC (Tab. 6.1). The tools themselves, and the insights obtained from their analysis, are outlined below. For brevity of exposition, we limit ourselves here only to the main insights that determined our tool-selection process. An in-depth discussion of all aspects learned from this comparison is, we believe, out of the scope of this thesis.

SonarQube[2]: SonarQube is a professional tool dedicated to the continuous analysis and measurement of source code quality. It has interesting features that relate to the detection of coding rules and is able to find possible bugs and violations. However, SonarQube is not simple to configure and execute. To do this, one has to edit several files to set parameters properly, and there is also a need of running a server to execute it.

SonarQube was designed to perform continuous inspection, which means that, as developers push their changes to a SCM (Software Configuration Management) system, a module triggers an automatic build and the project is analyzed, generating a report that is uploaded to a report server. It is possible to skip this step via command line to bypass the SCM. However, the data will still be stored at the server, and it requires executing queries to retrieve it. A tool with the option to generate a textual output would be a simpler and more effective alternative for our context. The performance achieved was not ideal. For instance, this tool took over 20 seconds to analyze a 6KLOC project. Moreover, SonarQube extracts only project level metrics, which does not comply with our fine-grained requirement.

---

2 http://www.sonarqube.org/

Google CodePro Analytix[3]: Like SonarQube, Analytix does not only compute software metrics, but also has features related to code analysis and software testing. It is possible to tune its behavior by disabling some features to increase performance, and by focusing only on selecting a set of metrics to be extracted. However, to gauge the maximal extent of the tool, we enabled all possible software metrics in our tests. Once properly configured, it is very simple to analyze a project with Analytix – all extraction configuration options can be summarized in three simple script files to be added to the project's root directory, followed by running a shell script on these files.

Regarding speed, our tests indicate that the tool's throughput (number of KLOC analyzed per second) decreases as project size increases. For example, a 3.6KLOC project was analyzed in 10 seconds; a 101 KLOC project, however, tool 23 seconds. Analytix also has a good documentation and delivers a satisfactory amount of metrics in a convenient way (XML and HTML reports). Analytix offers 22 project level-metrics, 25 package-level metrics, and 17 class-level metrics.

Analizo[4]: Analizo is a tool which can extract metrics from code in C, C++ and Java using the Doxygen third-party tool used for parsing code to automatically construct documentation[5]. We could successfully extract metrics from C++ code of single files using this tool. However, the tool has several bugs that make it unable to extract metrics from C++ source code directories in some cases. It was possible, however, to use the tool successfully on Java repositories. Analizo proved much slower than Analytix. For example, for a Java project of 26KLOC, Analizo took 70 seconds, whereas Analytix took only 17.3 seconds. In another test, Analizo tool 1440 seconds for a Java project of 101KLOC, whereas Analytix took only 23 seconds. Analizo can extract 10 project level metrics and 37 class level metrics.

iPlasma[6]: This tool was prominently advertised in the book of Lanza and Marinescu on object-oriented metrics extraction and interpretation [111], which is one of the key references in the field. The tool is supposed to handle C, C++, and Java code, and provide most of the object-oriented metrics described by the aforementioned book, which would more than cover our requirement for a rich set of fine-grained metrics. However, upon practical testing, we concluded that the tool has not been maintained for a long period of time, and as such it lacks a good and automated support of recent dialects of the above-mentioned languages. While the tool can be used in batch mode, its documentation is incomplete.

CCCC[7]: This tool has been used by various researchers to extract metrics from software repositories with the purpose of visualizing these [154]. Its main attrac-

---

3 http://marketplace.eclipse.org/content/codepro-analytix
4 http://www.analizo.org/
5 heep://www.doxygen.org
6 http://loose.upt.ro/reengineering/research/iplasma
7 http://cccc.sourceforge.net/

tive points are ease of use via the command line, zero-configuration effort (the tool only requests a root directory to analyze), and support of various programming languages, among which also C++ and Java. However, the tool is based on a lightweight parser that can handle relatively simple source code constructs, but easily gets confused by more complex ones, especially in C++. The error recovery mechanisms CCCC has are minimal, meaning that, upon certain errors, it will completely skip the remainder of the current file. As such, CCCC often generates a large amount of missing metric values.

SourceMeter[8]**:** This tool is a heavyweight extractor that aims to analyze source code in detail to find a wide variety of metrics, similar in spirit to SonarQube. We found SourceMeter to be quite hard to configure, and definitely lacking the automatic configuration we require in our context. We could only extract metrics from the project provided with the tool[9], for which all configuration files were already provided. Adapting these configuration files for other projects was far from trivial. For the above-mentioned sample project, the extraction tool 218 seconds on 14KLOC. This throughput is far too low for our context.

SciTools Understand[10]**:** Understand is a commercial tool for static code analysis. It provides an IDE which combines software testing, quality metric extraction, charts and visualizations of the relations and structures contained within the software project. There is a very comprehensive documentation available, with a very good support for usage and installation.

The tool offers a command line executable for batch execution, and the resulting metrics can be exported to a text format. The metrics can be extracted on several levels of granularity: 39 package level metrics, 43 class level metrics and 19 method level metrics. This tool achieves the best throughput among all considered tools we tested.

Summarizing the above observations, the best candidates found were Understand and CodePro Analytix. To gain more insight in these two tools, we compared them by extracting metrics from 9 open source projects. Table 6.1 shows the absolute and relative performances of the two tools. We see that Understand has best performance in most cases. Moreover, Understand extracts about three times more class-level metrics than Analytix. This, as noted earlier, is an important requirement for our goal to visually analyze high-dimensional metric datasets. Overall, Understand proved to satisfy all our requirements very well, so was selected to work further with.

---

8 http://www.sourcemeter.com
9 https://github.com/log4cplus/log4cplus
10 http://scitools.com/

Table 6.1: Performance comparison between CodePro Analytix and Understand.

Project Name	Analytix / Understand (KLOC/second)	Time Analytix (seconds)	Time Understand (seconds)
JMeter	101 / 118	23	30
Checkstyle	94 / 95	32	32
Gitblit	77 / 77	31	20
JUnit	26 / 26	17	10
JavaGame	3 / 3	10	4
Netty	116 / 194	70	66
Guava	76 / 242	120	168
Zxing	41 / 42	20	11
MPAndroidChart	20 / 20	14	8

6.4 Construction of Metric Evolution Maps

The previous section has explained how we extract a dynamic multidimensional dataset from a software repository. In detail, if we use Understand as a metric extractor, for a revision $r_t \in [r_s, r_e]$, our metric extraction delivers a set $E_t = \{e_i\}_{1 \leqslant i \leqslant N}$, where e_i are software entities in revision r_t. For Understand, these can be packages, files, and classes. We choose next to consider classes only, since these give the highest level of granularity of our analysis, *i.e.*, largest number of observations. If desired, packages and files can be handled analogously, with no change to the designs we present next. For classes, Understand delivers 43 metrics per class, i.e., e_i has $n = 43$ attributes.

To construct a visualization of this dynamic multidimensional datasets, two elements have to be considered – the visualization of a single revision, and the visualization of evolution, or dynamics, between revisions. The first aspect is detailed in Sec. 6.4.1. The second aspect is detailed in Sec. 6.4.2.

6.4.1 Revision Visualization

As outlined at the beginning of this chapter, we aim to construct a visualization which, for a given software revision, highlights the similarity of its entities. For this task, and following the discussions and results shown in the earlier chapters of this thesis, it is natural to choose a Multidimensional Projection (MP) to create a simpler representation of this data. For our case, such a projection will reduce the 43-dimensional set $E_t = \{e_i\}_{1 \leqslant i \leqslant N}$ to a two-dimensional set $P_t = \{q_i\}_{1 \leqslant i \leqslant N}$, so that similarities between classes $e_i \in E_t$ will be reflected by similarity in point positions $q_i \in P_t$.

One key difference with respect to the use of projections demonstrated in earlier chapters is that we have to consider the *dynamic* aspect when creating such

a projection. In detail: If we wanted to analyze a *single*, isolated, revision r_t, we could construct P_t by using any MP technique that preserves distances from the original 43-dimensional data space, such as, for instance, LAMP [95]. However, if we do that for another revision $r_{t'}$, there is no guarantee that we can next compare the projections P_t and $P_{t'}$ to reason about change. In other words, elements close in P_t will, indeed, denote similar classes in r_t, and elements close in $P_{t'}$ will denote similar classes in $r_{t'}$, respectively. However, elements in P_t close to elements in $P_{t'}$ do *not* necessarily denote classes in the two revisions which are similar (possible false positives can appear). Similarly, classes in the two revisions which are indeed similar may not project to elements close in P_t and $P_{t'}$ (possible false negatives can appear). This problem is inherent for any projection technique that treats revisions independently.

As such, when constructing the visualization for a single revision using a projection technique, we must take into account the other revisions too. Intuitively, we want that, for revisions r_t and $r_{t'}$ which are close in time ($|t - t^{prime}|$ is small), classes from the two that are similar project in areas of the 2D embedding space which are close to each other. In other words, in areas where the data does not change much, the projection should not change much. On a high level, this will preserve the 'mental map' of the users who visualizes the software evolution, allowing them to associate stability in the projection with stability in the data, and change in the projection with change in the data, respectively. As such, we say that a suitable projection for our dynamic multidimensional dataset should exhibit two kinds of coherence:

- *spatial* coherence, *i.e.*, the fact that similar observations in the same revision r_t should project to close points in P_t;

- *temporal* coherence, *i.e.*, the fact that similar observations in revisions r_t and $r_{t'}$ which are close in time should project to points which are close in P_t and $P_{t'}$.

Ideally, a projection technique should achieve both above types of coherence. However, achieving this in practice is challenging, for the reasons discussed under MDS in Sec. 6.2. Recently, a technique called dynamic t-SNE technique (dt-SNE) was proposed, that achieves precisely that [151]. To briefly outline how this is done, let us the cost function minimized by the original t-SNE technique [193] that dt-SNE extends:

$$C_{t-SNE}[t] = \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \mathcal{P}(\mathbf{e}_i[t], \mathbf{e}_j[t]) \log \left[\frac{\mathcal{P}(\mathbf{e}_i[t], \mathbf{e}_j[t])}{\mathcal{P}(\mathbf{q}_i[t], \mathbf{q}_j[t])} \right] \qquad (6.1)$$

In the above, $\mathbf{e}_i[t]$ denote observations at time step, or moment, t, and $\mathbf{q}_i[t]$ denote their respective projections. The term $\mathcal{P}$ can be considered as a random process whose value is high when two entities (observations or projections) are near. This cost function corresponds to the Kullback-Leibler divergence between

$\mathcal{P}(\mathbf{e_i}[t],\mathbf{e_j}[t])$ and $\mathcal{P}(\mathbf{q_i}[t],\mathbf{q_j}[t])$, which penalizes neighbors in E_t to be far apart in P_t – that is, it penalizes the appearance of missing neighbors in P_t.

To incorporate temporal coherence, dt-SNE extends the above cost to

$$C_{dt-SNE} = \sum_{t=r_s}^{r_e} C_{t-SNE}[t] + \frac{\lambda}{2N} \sum_{i=1}^{N} \sum_{t=r_s}^{r_e-1} \|\mathbf{q_i}[t] - \mathbf{q_i}[t+1]\|^2 . \qquad (6.2)$$

Here, $\|\cdot\|$ denotes Euclidean distance in 2D. The added cost term in Eqn. 6.2 penalizes projections of observations from moving too much between consecutive time moments. It avoids the problem of abrupt changes between consecutive projections, but keeps the property of adjusting the layout to reflect similarity changes among its observations. The parameter λ controls the trade-off between preserving spatial coherence (like t-SNE does) and preserving temporal coherence. Small values of λ favor the former, while larger values favor the latter. For further details on dt-SNE, we refer to [151].

Given the above desirable coherence properties, we adopted dt-SNE as a projection technique to create a sequence of 2D projections P_t from our sequence of multidimensional metric datasets E_t. Next, we use the visual explanatory techniques described in Chapter 5 to explain the projections P_t in terms of colored regions, outlined clusters, cluster labels, and dimension-value bars. However, we adapted the dimension-ranking color mapping so as to consider the fact that we now color map an entire set of projections rather than a single one. In detail, we assign colors to the most frequent top-ranked dimensions throughout *all* revisions $r_t \in [r_t, r_e]$. This way, a top-ranked dimension is mapped by the same color throughout the entire set of visualizations P_t.

6.4.2   Evolution Visualization

The techniques presented in Sec. 6.4.1 let us visualize a single revision from a repository by a colored and annotated projection. As already mentioned in Sec. 6.4.1, dt-SNE preserves well *both* (a) distances between points in the same projection and (b) between points in projections for close time-frames. Feature (a) enables us to reason about groups of similar classes at any given time, *i.e.* supports answering **Q1**. Feature (b) allows us to reason about the amount of change (of class metrics) by looking at the amount of visual change, *i.e.* supports answering **Q2**.

To visualize change, we propose two options. First, we can use a small-multiple or animation of the projections P_t, drawn all using the same metric-ID-to-color mapping, see *e.g.* Fig. 6.2 (for more details, see Sec. 6.5.1). These techniques are good for analyzing a short time-range or getting a coarse overview of the dynamics in an entire project. However, for hundreds of revisions, such an approach cannot show detailed insights.

For this, we propose a second method: For each class $\mathbf{e}_i$, let $\pi_i = (\mathbf{q}_i^{r_s}, \ldots, \mathbf{q}_i^{r_e})$ be its 2D projections in $P_{r_s}, \ldots, P_{r_e}$. We next construct polylines from the points π_i for all classes. These can be thought of as 'trails' indicating the evolution of classes. For projects with hundreds of classes and revisions and significant change dynamics, drawing the raw trail-set $\{\pi_i\}$ creates a cluttered image. Instead, we show a simplified view by bundling trails using an existing high-performance edge-bundling technique [194] (any other trail or general-graph bundling techniques can be used equally well). Figure 6.1a shows an example for the Guice repository evolution (described in detail in Sec. 6.5.2). Trails are color-coded from green (first revision r_s) to red (last revision r_e), similar to edge color-coding in other software visualizations [83]. The trail pattern allows reasoning about change dynamics: Short trails indicate classes that stay very similar to each other, metric-wise. Long trails indicate classes having large changes. Bundle splitting and merging show classes that become more different, respectively more similar, during the project lifetime.

This type of image gives a high-level *overview* of the project dynamics. For *detail* insights, we provide two additional views: First, we can select a specific group of classes (in any time-frame P_t) and explain their entire evolution from a metrics perspective (Fig. 6.1b). Here, all trails that do not pass through the selected classes are rendered gray (to provide context); a trail that contains a selected class $\mathbf{e}_i$ is rendered, at each point $\mathbf{q}_i^t$ with the color mapping the top-ranked metric for revision r_t for the cluster containing $\mathbf{e}_i$ at that moment. Trails change colors, thus, showing how the most important metrics explaining the similarity of a class at each moment of its evolution changed. In Fig. 6.1b, for instance, we see that most of the selected trails are orange, which maps to the *average essential cyclomatic complexity* metric (as encoded by the top-right color legend).

In addition to the above, we can also select a revision of interest r_t, and show the evolution of all classes around this moment (Fig. 6.1c. For this, we show only trail fragments in the interval $[t - \delta, t + \delta]$, where δ is the user-specified size of the window of interest centered at t, typically a few revisions. Fragments are alpha blended with a Gaussian profile centered at t and vanishing at the window borders, so as to focus the visualization on the moment of interest, and are colored like described earlier for Fig. 6.1b. Overlaying these trails atop of the projection P_t shows the local dynamics of each class, *i.e.*, where it came from, and where it will go next, from the perspective of revision r_t.

6.5 Sample applications

We next illustrate our approach to visualizing dynamic multidimensional metric datasets, which we call *metric evolution maps* (MEMs), by analyzing two well-known open source repositories: JUnit [97] and Google Guice [68]. For both repositories, we selected 100 revisions and extracted 43 metrics per class, using the approach outlined in Sec. 6.3.2. To focus on the most salient time dynamics, we
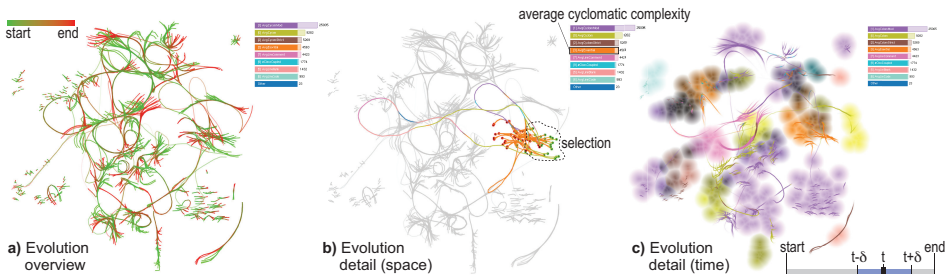
Figure 6.1: Evolution visualization. (a) Overview showing changes of all classes in all revisions. (b) Evolution of selected classes color-coded by top-ranked metrics. (c) Evolution of all classes around a selected revision.

kept only classes that are present in all considered revisions. Next, we created the respective projections using dynamic t-SNE, as explained in Sec. 6.4.1, and finally explored these interactively.

6.5.1 JUnit 4

JUnit is a popular Java testing framework. It allows unit testing by providing helper classes which repeatedly invoke methods from the tested class and compare with expected results. We considered here revisions from March 2010 to March 2016. Data acquisition delivered us 314 classes. Per revision, JUnit has 20.5 KLOC on average.

Figure 6.2a-d shows an overview of the start phase of the evolution (revisions 1-4), using small multiples. We see that the overall cluster layout is kept well by dt-SNE, which is indeed expected for a relatively small time-span of 4 revisions. This helps when comparing consecutive maps. Let us now compare the first with the last (100th) revision (Figs.6.2e,f). While the spatial distribution of clusters changes considerably (which is expected, given the 6 year period being studied), we see that the main clusters of similar classes are caused by the *same* metrics – see the similarity of the color legends in Figs. 6.2e,f, both in terms of bar colors and bar order. For instance, most classes in both revisions 1 and 100 are similar due to the metrics *number of default methods* (yellow) and *average cyclomatic complexity* (purple). This means that, even though *individual* classes do change, there is a strong underlying grouping of classes in terms of aspects captured by the above metrics. We also see, in all Figs. 6.2, an isolated stable outlier orange group, explained by the metric *average cyclomatic modified*. While we did not dig deeper into the semantics of this class-set, it is clear that they evolve (or rather, stay constant) in a very different way than the rest of the code.

The evolution trails for the entire dataset show a high dynamics (Fig. 6.2g). We see here several isolated, coherent, bundles. These indicate classes which changed their metric values *together*, *i.e.*, evolved as a 'block'. One instance is the bottom
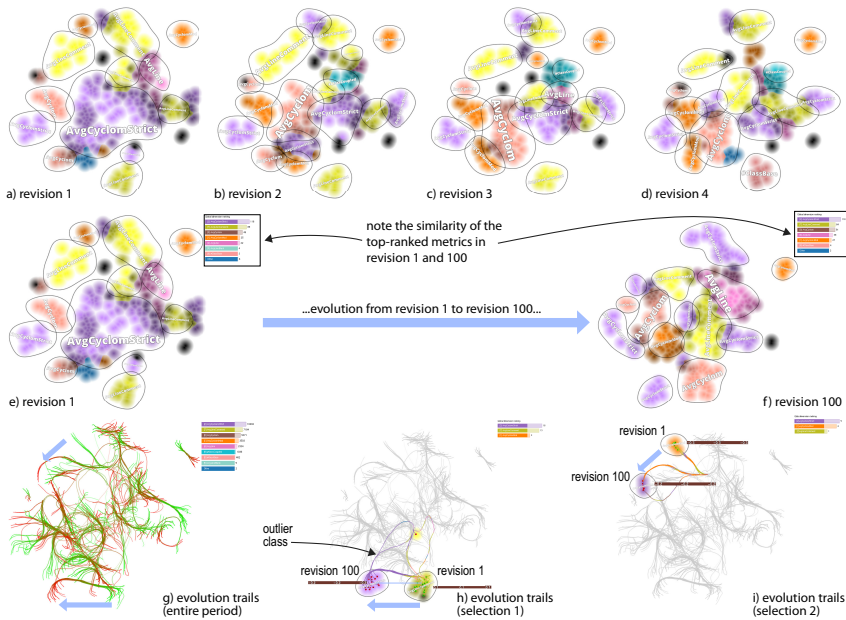
Figure 6.2: Metric evolution maps for four revisions of the JUnit project. See Sec. 6.5.1.

horizontal bundle in Fig. 6.2 which moved in the indicated arrow sense. To understand this better, we select one of the end clusters of this bundle and show evolution details (Fig. 6.2h). The selected group (16 classes) are related mainly by *average cyclomatic complexity* (yellow) in revision 1, and evolved together, being finally mainly similar due to *number of default methods* (purple). Brushing the views shows us that these classes belong to packages *org.junit.internal.runners* and *org.junit.internal.builder*. A stray trail (purple, Fig. 6.2h, shows a single class diverging from the grouped change. Fig. 6.2i shows a second group-analysis example. The selected classes change together (as told by the bundles), but due to several metrics (as told by the bundle colors): First, this group was explained by the metric *average cyclomatic codified* (orange), next by *average line comment* (yellow), and next by *average strict cyclomatic complexity* (purple).

6.5.2   Google Guice

Google Guice is a Java framework which provides dependency injection for Java objects [68]. Here, we analyze how 524 classes evolve during 100 revisions between April 2007 and March 2016. The maps of revision 1 and 100 show that most classes share similar values of the metrics *average modified cyclomatic complexity* (purple) and *average cyclomatic complexity* (yellow) (Figs. 6.3a,b). The overview trails (Fig. 6.3c) show, in contrast to the JUnit repository (Sec.6.5.1), many more short trails. This tells that Guice has many more stable classes, which

do not change much with respect to other classes. The color legend in Fig. 6.3c tells that the values of the *average modified cyclomatic complexity* metric are the most similar for most classes during the analyzed period.

As the images show, the largest group in revision 1 is explained by metric *average strict cyclomatic complexity* (purple) splits during the evolution. To further study this, we use the animation of the projections (see Sec. 6.4.2) to find out that splitting occurs at revision 22. This is also visible in the revision-centric visualization in Fig. 6.3d. During this process, the splitting bulk of purple classes is joined by a distinct group of yellow classes, explained by the metric *average cyclomatic complexity*.

We next focus on the question of what explains the change of class-groups having a high dynamics. To study this, we select a group of classes showing high dynamics, *i.e.*, part of long trails (see next Fig. 6.3e). This group has the interesting behavior of moving very close to its original position on the last revision (100). As such, we want to understand which were its intermediate values during its evolution. After selecting the group classes, we noticed a prominent region of blue colored edge fragments half-way of the group's trajectory. According to the color legends, blue can map the metrics *count of coupled classes* or *average number of lines of code*, so we decided to refine the investigation. By displaying the windowed trails (see Sec. 6.4.2), we found that the blue edge fragments appear around the 24th revision. Brushing the metric values, we confirmed that the most similar metric values in revision 24 are, indeed, the *count of coupled classes* and *average number of lines of code*.

6.6 Discussion

We next discuss the main aspects of our metric evolution maps (MEMs).

Advantages: Our method is easy to use, computationally efficient (runs in real-time for datasets up to 10K observations on a typical PC for a C++ CPU single-threaded implementation having a recent NVidia card), and generic (can be used on any set of artifacts, as long as one has a way to extract several quantitative metrics on these artifacts). MEMs create partitions of projections of the available data entities (classes, files, packages) in terms of groups of entities that are most similar from the perspective of any of the underlying metrics. Most importantly, users do not have to *select* which these metrics are – they are determined automatically by the visualization. The resulting segregations of projections in similar groups can be displayed either *implicitly, i.e.,* as a colored image consisting of several same-color zones; or *explicitly, i.e.,* as disjoint point clusters. The implicit view is useful when one requires a high-level overview of the entire projection 'landscape', such as in presentations. The explicit view is useful when one needs to reason in more depth, and about specific groups of points in the projection.
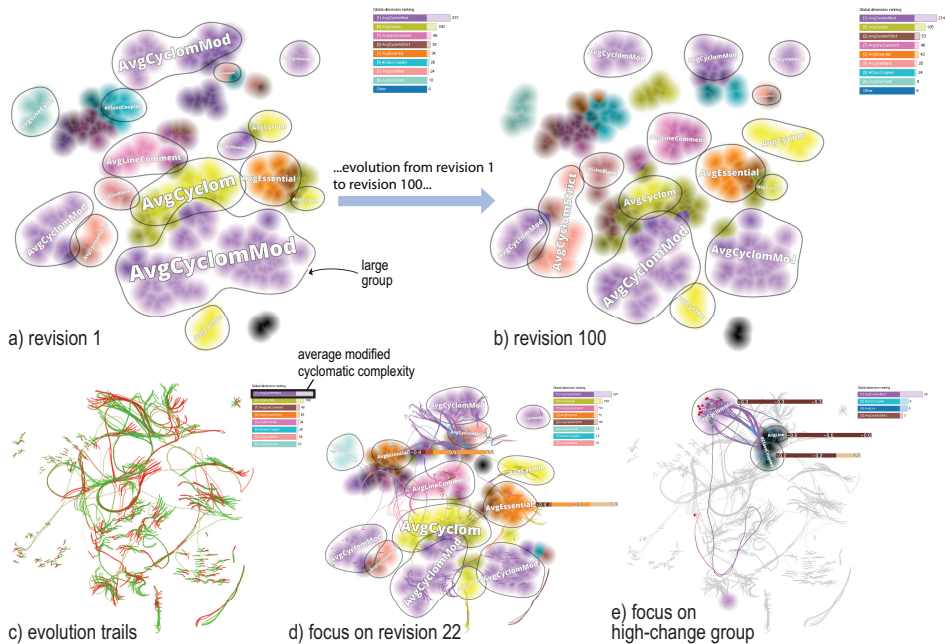
Figure 6.3: Guice repository. (a,b) First and last revisions. (c) Evolution trails, entire period. (d) Focus on revision 22. See Sec. 6.5.2.

By employing a MP technique crafted for time-dependent data, dynamic t-SNE, MEMs can explain how groups of software artifacts are similar to each other at any time moment (revision), and also how they change across time. The key advantage of using MPs is the high visual scalability thereof: Datasets are essentially reduced to scatterplots, irrespective of the number of underlying dimensions (metrics). Using a high-quality projection, such as dt-SNE, guarantees next the accuracy of the obtained visualizations. Change visualization is supported by a simplified view, consisting of bundled trails. While bundling does remove small-scale spatial-change details, it also massively reduces clutter, letting users perceive the main motions (changes) of highly-related entities. Since absolute 2D position in a projection does not have a formal meaning, we believe this simplification has strong advantages. This is especially true for the t-SNE class of projections, which do not optimize for distance preservation, but neighborhood preservation. Finally, color coding is used to explain which metrics are the most important to explain the change.

Parameters: MEMs have only three free parameters which are intuitive and simple to control: ρ, ρ_c, and α, which have been detailed in the context of the attribute-based visual explanation of projections (Chapter 5. Apart from these parameters, the dynamic aspect introduced by MEMs adds a single parameter λ (Eqn. 6.2) which controls the trade-off between spatial and temporal coherence.

Setting $\lambda = 0.1$ has given an empirically assessed good balance between the two. The partition of a projection into regions, which are next explained by means of color-coding, outlines, dimension labels, and dimension value bars, is fully automatic. The only interaction required here is brushing regions to bring up the dimension value bars.

Limitations: MEMs inherit several limitations from the underlying attribute-based visual explanation of projections, as follows. Color-coding explanations are inherently limited to the maximum number of colors that a categorical colormap can reasonably have (about 10 colors). The current similarity-ranking metrics (Sec. 5.2) are, in general, not well suited to work very high-dimensional datasets (beyond roughly 50 dimensions).

Apart from these, MEMs have some limitations of their own. The current visual designs we proposed here are geared towards showing the evolution of entities that exist through the entire evolution period. While, technically speaking, we could use our proposal to show entities that have shorter lifespans, such as classes which are created and/or disappear during the project lifetime, future work is needed to emphasize the precise lifetimes of such entities.

Arguably the largest limitation of the presented techniques relates to their perceived value in terms of connecting to concrete end-user problems such as software maintenance tasks. We acknowledge not having studied these aspects. However, *prior* to being able to address such issues, we have the technical challenge of being able to show the dynamics of large sets of entities, as captured by large sets of quality metrics. We believe that our current work has addressed this technical challenge up to a large extent. Using our techniques to concretely address actual real-world maintenance problems is a key task, but one for future work.

6.7   Conclusion

We have presented a set of techniques that allow users to explore the evolution of entities in software repositories from a metric-centric perspective. For this, we adapt and extend the visual explanations proposed in Chapter 5 for static multidimensional data to handle time-dependent multidimensional datasets. Key to our technical proposal is the ability of visually explaining groups formed in multidimensional projections, and the evolution in time of (parts of) these groups, in terms of individual entity attributes, such as software metrics. Our techniques include a mix of annotations and segmentations of static 2D projections, bundle-based and metric-based simplified visualizations of the evolution of entities in time, and several interactive mechanisms to provide level-or-detail insight on selected entities and metrics. We demonstrate the technical applicability of the proposed techniques on two real-world software repositories.

Future work can target several directions. First, the proposed visualizations can be enhanced to show additional metrics, such as the speed of change, identity of

changers, and identity of the changed artifacts, in line with earlier repository exploration methods [199, 154]. Secondly, researching scalability in terms of number of metrics can lead to novel visualization methods. Thirdly, validating the end-to-end added-value of the proposed exploration methods should be done by organizing actual user studies involving concrete maintenance tasks. Beyond the scope of software visualization, our methods could be tested and applied to the more general challenge of understanding any multidimensonal time-dependent dataset. Finally, it is interesting to consider how to combine the bundling technique used here to map trails of evolving entities with the similarity-driven bundling proposed in Chapter 4. Doing so would add more semantics to our bundled trails, in terms of grouping sets of related entities in an even stronger way.

Discussion and Conclusions

7

In this thesis, we have presented several methods for the visual exploration of similarity-induced relations in the context of multidimensional datasets. Specifically, in Chapter 3, we have shown how to make similarity tree more scalable, and display richer information. Chapter 4 showed how similarity trees can be used to add semantics, in terms of observation similarity, to edge bundling displays of multidimensional relational data. Chapter 5 has proposed methods for the visual explanation of spatially compact regions in a multidimensional projection from the perspective of dimensions that make observations similar. Chapter 6 has extended these explanations to time-dependent multidimensional datasets.

In this final chapter we revisit our initial research questions, stated in Section 1.3, discussing our results and reasoning about how well they cover the problems concerning these questions. We conclude this discussion by highlighting promising directions for future research based on the results presented so far.

To start with, let us recall the first research question that motivated this work:

RQ1: How can we enhance similarity trees so they become computationally scalable and also provide local and automatic explanations of their subtrees?

We addressed this question at two different levels. In Chapter 3, we proposed multiscale implementation of a high precision similarity tree for exploration of multidimensional data, the Visual SuperTree (VST). The VST scales better, both computationally and visually, than standard similarity trees, by working with a multiscale partitioning of the input multidimensional data. By building a set of trees, one for each scale of this partitioning, the VST can process large datasets at competitive precision and computational times. The visualization of such data is next possible using a multiscale approach: One can visualize either a uniformly simplified representation of the entire dataset (by selecting a level of the aforementioned multiscale partitioning), or a mixed-level representation, where certain regions of the data are summarized and other regions are shown in more detail (by interactively collapsing or expanding parts of the current VST). The examples presented in Chapter 3 show that the VST can handle datasets of millions of entities and hundreds of dimensions.

Next, in Chapter 4, we used the VST in the context of simplified visualization of relational datasets, using bundling techniques. The Similarity-driven Edge Bundling (SDEB) uses as similarity backbone the hierarchy of the VST to control how relations between the input observations are aggregated. Differently than

most existing bundling techniques, which only use the nodes' geometric informa-
tion to group edges, SDEB creates bundle patterns based on the similarities from
node attributes. This way, SDEB generalizes recent work in attribute-driven edge
bundling. Such relations can either be provided explicitly, as in a classical graph
dataset, but can also be computed from the similarities of observations (nodes)
themselves. As such, SDEB bridges the domains of multidimensional data visual-
ization and relational data visualization. Similarly to VSTs, SDEB offers a multi-
scale way to explore the simplified (bundled) representation, which generalizes
earlier multiscale exploration methods such as hierarchical edge bundling (HEB)
to the case where one does not have an explicit hierarchy.

Both VST and SDEB approaches offer automatic summarizations of their sub-
trees. The VST automatically builds image and text mosaics, thus allowing a com-
pact and intuitive presentation of the type of observations contained in a group of
similar observations. SDEB creates tag clouds when dealing with associated tex-
tual data, thus allowing a quick inspection of the main topics covered by groups of
similar nodes. Such summarizations are, in our context, a first step of explaining
what kind of entities constitute a group of related entities.

We now recall our second research question:

RQ2: How can we enhance multidimensional projections with explanatory mech-
anisms that provide local and automatic explanations of the perceived pat-
terns?

Note that both RQ1 and RQ2 revolve around the notion of *similarity* in mul-
tidimensional datasets, and how this can be explained to users. The main differ-
ence between the two questions is that we approach RQ1 focusing on one specific
technique, similarity trees, and RQ2 by focusing on another specific technique,
respectively multidimensional projections. We approache RQ2 by considering the
cases of static (time independent) and dynamic (time dependent) multidimen-
sional data visualized with projections.

For static data, we showed in Chapter 5 how a projection can be explained
locally in terms of the most prominent dimensions that make points in a neigh-
borhood similar to each other. Compared to global explanations, such as biplot
axes, axis legends, and color coding, our explanations have the key added value
that they can freely *vary* across a projection – thus, they locally adapt to providing
the best way (within the reach of our proposed metrics) to tell why points are sim-
ilar. This implicitly segments a projection, even in the case that it does not contain
a set of visually well-separated clusters, into regions that have different explana-
tions. If desired, we showed how we can make the separation, or segmentation,
of a projection into distinct clusters explicit, and how these clusters can be next
explained by means of the same small set of dimensions that contribute most to
their points' similarity. Compared to other explanatory techniques for projections,
we believe that our proposal is simpler to understand and use, as it produces a

visual map consisting of regions having different colors and textual annotations, a metaphor which should be familiar to most users.

In Chapter 6, we extended the visual explanations proposed in Chapter 5 to the context of dynamic multidimensional datasets. For this, we used dynamic t-SNE, a very recent technique that aims to preserve both spatial and temporal coherence in such datasets. As a collateral added value, our application of dynamic t-SNE to explore such datasets is the first real-world usage of this technique on large and complex datasets emerging from a concrete application area – in our case, program understanding. This chapter also presents additional material that illustrates the way of working and added-value of the visual explanatory techniques introduced in Chapter 5 for real-world datasets. Additionally, Chapter 6 introduces a novel way to visualize change in dynamic multidimensional datasets by means of bundled trails of observations. Trail visualization provides a compact, visually scalable, summarized representation of the evolution in time of hundreds up to thousands of observations across tens up to hundreds of time frames. Additional color coding and interaction allows inspecting which groups of similar observations change in time, and why. We demonstrated our approach by presenting the construction of an end-to-end pipeline for mining and visualization of evolving software quality metrics from large software repositories.

7.1 Advantages and Limitations

It is interesting to reflect on the various advantages and limitations of the proposed techniques from a high-level perspective – that is, beyond the discussions provided in the concluding remarks of each chapter. This is also the moment of reflecting on comparing the approaches proposed by these techniques to the high-level aim of visually encoding similarity in multidimensional data.

Our proposed approaches essentially visualize the same information – similarity of multidimensional observations. However, how and which similar observations are selected, and how is similarity visualized, differs. VSTs select similar entities based on an explicit clustering process working in high-dimensional data, and encode these in the node-link structure of the visualized similarity tree. Spatial positioning of the tree nodes is driven by this tree structure, and not the explicit similarities of the data elements, and is thus subject to placement decisions taken by tree layout algorithms. VSTs make explicit the level of similarity of different groups of items, by encoding this into the paths between such groups, and their lengths. SDEB acts very much in the same way. Its main difference with respect to VSTs is that it can handle also association relations between observations, by grouping associations between similar observations into the same bundle(s). Projection-based methods work differently. First, they select similar entities based on visual proximity, based on the assumption that the underlying projection can preserve distances well. This can be seen as an implicit or explicit clustering process working in the low-dimensional projection space. Spatial positioning of

observations is driven fully by the projection, rather than by some clustering technique based on data. However, we can argue that projection explanations are subject to the quality of the underlying projection techniques in the same way that similarity-tree based explanations are subject to the quality of the underlying tree layout techniques. In this sense, all our explanatory mechanisms for similarity are only as good as the 'weakest link' in the entire data processing chain.

All presented techniques are designed to offer a high level of visual scalability. VSTs achieve this by using the multiscale representation provided by the underlying data clustering. SDEB achieves this both by leveraging the aforementioned multiscale, and also by exploiting the inherent visual scalability of edge bundling. Attribute-based explanations of projections achieve scalability by leveraging this property that all projection techniques have, and by using compact image-based techniques to explain groups of similar observations in the projection. Additionally, all our techniques have a visual multiscale component: VSTs and SDEB achieve this by allowing one to choose the level-of-detail at which a tree, respectively a set of bundled edges, is to be drawn. Attribute-based explanations of projections achieve this by allowing one to explain local groups by means of a single dimension or a dimension set. In other words, VSTs and SDEB propose a multiscale in the space of number of *observations*, whereas the attribute-based projection explanations propose a multiscale in the space of number of *dimensions*.

Preservation of the mental map during exploration of large datasets is also a shared concern for all our techniques. For VSTs, this is achieved by using minimaps to link the current and previous navigation views. For SDEB, this is achieved by keeping the position of leaf nodes fixed during the multiscale exploration. For the attribute-based exploration of dynamic datasets, this is achieved by choosing a suitable multidimensional projection technique (dt-SNE) that preserves temporal coherence, and by using consistent visual encodings across different time frames.

However, our proposed techniques also have several limitations. At a high level, we underline the following ones. VSTs and SDEB inherently rely on the quality of a clustering algorithm. As explained in Chapters 3 and 4, clustering results are significantly influenced by the choice of suitable parameters. Moreover, the binary division of a dataset in clusters may not be the best way to encode similarities in case of a relatively flat similarity distribution. Separately, VSTs and SDEB explain groups of similar entities by depicting representative samples or sample values. This works well when the dataset at hand allows an intuitive summarization in this sense. However, in other cases, the summarization proposed by the attribute-based projections, which works in terms of representative dimensions, may be better. It is interesting, thus, to consider a way to unify the two types of summarization. Separately also, projection methods are directly affected by the well-known curse of dimensionality problem. For very high dimensional datasets, the resulting projections may not be informative, and therefore searching for reasons, and visually explaining, why points are close to each other in such projections may be meaningless. In this direction, it is interesting to consider more advanced confidence metrics that assess the quality of a local explanation, and only show it if

it can provide a minimal degree of meaningfulness. Finally, for dynamic datasets, projections need to fight the trade-off between spatial and temporal coherence. Preserving both types of coherences well in a long and complex sequence of time-frames may be impossible. As such, it is interesting to consider ways to inform the user about the quality of both types of coherence preservation, *e.g.* by extending the proposals of Martins *et al.* [118] to dynamic datasets.

7.2 Future work directions

Based on the above discussion, a non-exhaustive list of extensions of the current research contains the following points:

Data variety: The techniques presented in this thesis are focused on quantitative multidimensional data. As described in Section 2.1.1, a wide variety of multidimensional data exist, including, among others, categorical and relational data. Providing means to explore this variety can allow a much wider scope of applications. In particular, as mentioned earlier, it is interesting to see how to use the image-and-text summarization options provided for VSTs and SDEB to annotate groups of similar points in a projection; and conversely, to provide the dimension-based explanation of projection groups to supernodes in the VST or SDEB. Beyond this, new summarization mechanisms for different types of data, such as sound and video, would lead to richer and easier usable depictions of big data.

Data volume: The explanation of multidimensional projections are built at fine-scale, *i.e.,* we explain the prominent dimensions of all projected points. This can impose computational limitations for the exploration of projections of hundreds of thousands, or millions of points, and/or projections of dynamic datasets having thousands of time steps. As such, we envisage that designing multiscale projection methods, which would extend the current idea of using landmarks or representatives [95, 168] to several levels of details, possibly using similarity-based clustering (like done for the VST construction), could make significant gains in handling large datasets.

Combining techniques: Our work can be summarized in two types of visual metaphors for showing similarity: trees and projections. Each has its own advantages and limitations, as discussed earlier. It is thus interesting to see whether the methods could be combined to keep these advantages and reduce limitations. For instance, we envisage using projections to place the leaves of a similarity tree more accurately with respect to their similarities, followed by a constrained layout of the remaining tree nodes, to serve for VST visualization. Conversely, we envisage augmenting a projection-based display for multidimensional data by a similarity-tree, to simplify the interpretation of attribute-based views of projec-

tions.

Evaluation: Although evaluation of our proposed techniques took place by using a (we believe) sufficiently rich collection of datasets of various provenances, types, and sizes, end-to-end evaluation of the *added value* that such techniques bring in a concrete application was limited. Important aspects such as usability, clarity on the visual metaphor, and ease of learning of the interactive tools, can only be accurately measured after performing a proper user evaluation. Additionally, measuring the delivered insight should involve domain experts using our tools and techniques to solve concrete problems.

Data analysis domains: Our proposed techniques are focused on the improvement of multidimensional projections and similarity trees to get insight into unsupervied data analysis, i.e., the most generic form of data analysis. However we acknowledge that discriminative visualizations of labeled datasets in supervised settings can be extremely useful. As such, we envisage refining our results for this more specific context.

Bibliography

[1] C. C. Aggarwal and C. X. Zhai. *Mining Text Data*. Springer, 2012.

[2] F. Alimoglu and E. Alpaydin. Combining multiple representations and classifiers for pen-based handwritten digit recognition. In *Proceedings of the 4th International Conference on Document Analysis and Recognition*, volume 2, pages 637–640, 1997.

[3] G. Antoniol. Detecting groups of co-changing files in CVS repositories. In *Proceedings of the 8th International Workshop on Principles of Software Evolution*, pages 23–32. IEEE Press, 2005.

[4] D. Archambault, H. C. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, 2011.

[5] H. G.-M. (Author), J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Pearson, 2 edition, 2008.

[6] K. Bache and M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data.

[7] C. Bachmaier, U. Brandes, and B. Schlieper. Drawing phylogenetic trees. In X. Deng and D. Du, editors, *Proceedings of the 16th international conference on Algorithms and Computation*, volume 3827, pages 1110–1121, 2005.

[8] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of ACM Symposium on Software Visualization*, pages 165–172, 2005.

[9] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. The state of the art in visualizing dynamic graphs. *EuroVis STAR*, 2014.

[10] R. Becker, W. Cleveland, and M. Shyu. Visual design and control of trellis display. *Journal of Computational and Graphical Statistics*, 5(5):123–155, 1996.

[11] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transaction on Graphics*, 21(4):833–854, 2002.

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

[13] E. Bertini, A. Tatu, and D. Keim. Quality metrics in high-dimensional data visualization: an overview and systematization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2203–2212, 2011.

[14] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[15] C. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2007.

[16] Q. W. Bouts and B. Speckmann. Clustered edge routing. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 55–62, April 2015.

[17] U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In M. Kaufmann and D. Wagner, editors, *Proceedings of the 14th International Conference on Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2007.

[18] B. Broeksema, A. Telea, and T. Baudel. Visual analysis of multidimensional categorical data sets. *Computer Graphics Forum*, 32(8):158–169, 2013.

[19] C. Buchheim, M. Jünger, and S. Leipert. Improving walkers algorithm to run in linear time. In *Graph Drawing*, pages 344–353. Springer, 2002.

[20] M. Burch and S. Diehl. TimeRadarTrees: Visualizing dynamic compound digraphs. *Computer Graphics Forum*, 27(3):823–830, 2008.

[21] H. Byelas and A. Telea. Visualization of areas of interest on software architecture diagrams. In *Proc. ACM SoftVis*, pages 105–114, 2006.

[22] H. Byelas and A. Telea. Visualizing metrics on areas of interest in software architecture diagrams. In *2009 IEEE Pacific Visualization Symposium (PacificVis)*, pages 33–40, 2009.

[23] C. Catlett, W. Gentzsch, L. Grandinetti, G. Joubert, and J. Vazquez-Poletti. *Cloud Computing and Big Data*. Advances in Parallel Computing. IOS Press, 2013.

[24] S. Chacon. *Pro Git*. Apress, Berkely, CA, USA, 1st edition, 2009. ISBN 1430218339, 9781430218333.

[25] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proceedings of the 7th Conference on Visualization*, pages 127–131, San Francisco, CA, USA, 1996.

[26] T. Chan and L. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266–277, 2001.

[27] A. Cockburn, A. K. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*, 41(1):1–31, 2008.

[28] D. Coimbra, R. Martins, A. Telea, and F. Paulovich. Explaining 3d dimensionality reduction plots. *Information Visualization*, 15:154–172, 2015.

[29] D. B. Coimbra. *Multidimensional projections for the exploration of multimedia data*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Groningen, the Netherlands, April 2016.

[30] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[31] K. A. Cook and J. J. Thomas. Illuminating the path: The research and development agenda for visual analytics. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), 2005.

[32] P. Cortez and A. Morais. A data mining approach to predict forest fires using meteorological data. In *Proceedings of the 13th Portuguese Conference on Artificial Intelligence*, pages 512–523, 2007.

[33] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547 – 553, 2009.

[34] A. M. Cuadros, F. V. Paulovich, R. Minghim, and G. P. Telles. Point placement by phylogenetic trees and its application for visual analysis of document collections. In *IEEE Symposium on Visual Analytics Science and Technology (VAST 2007)*, pages 99–106, Sacramento, CA, USA, 2007.

[35] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, 2008.

[36] R. R. O. da Silva, P. Rauber, R. M. Martins, R. Minghim, and A. Telea. Attribute-based visual explanation of multidimensional projections. In *Proceedings of the 2015 EuroVis Workshop on Visual Analytics (EuroVA 2015)*, pages 137–142, 2015.

[37] R. R. O. da Silva, J. G. de Souza Paiva, G. P. Telles, F. Rolli, C. E. A. Zampieri, and R. Minghim. The Visual Supertree: Similarity-based multiscale visualization. Submitted to Information Visualization (IVI), 2016.

[38] R. R. O. da Silva, P. E. Rauber, and A. C. Telea. Beyond the third dimension: Visualizing high-dimensional data with projections. *IEEE Computing in Science & Engineering*, 2016. Accepted for publication.

[39] R. R. O. da Silva, E. F. Vernier, P. E. Rauber, J. L. D. Comba, R. Minghim, and A. Telea. Metric Evolution Maps: Multidimensional attribute-driven exploration of software repositories. Accepted for publication in the 21st International Symposium on Vision, Modeling and Visualization (VMV)., 2016.

[40] M. J. J. de Hoon, S. Imoto, J. Nolan, and S. Miyano. Open source clustering software. *Bioinformatics*, 19:1453–1454, 2004.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[42] T. Dharani and I. L. Aroquiaraj. A survey on content based image retrieval. In *Proceedings of the International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME)*, pages 485–490. IEEE, 2013.

[43] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ, 1999.

[44] S. Diehl. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, Berlin, 2008.

[45] S. Diehl and A. C. Telea. Multivariate networks in software engineering. In A. Kerren, H. C. Purchase, and M. O. Ward, editors, *Multivariate Network Visualization – Dagstuhl Seminar #13201, 2013*, volume 8380 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2014.

[46] S. Ding, H. Zhu, W. Jia, and C. Su. A survey on feature extraction for pattern recognition. *Artificial Intelligence Review*, 37(3):169–180, 2012.

[47] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 10(55):78–87, 2012.

[48] J. Dong and D. Fernández-Baca. Construction large conservative supertrees. In *Proceedings of the 11th International Workshop on Algorithms in Bioinformatics (WABI)*, LNBI 6833, pages 61–72, 2011.

[49] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Proc. ICSM*, pages 109–118, 1999.

[50] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Cybernetics and Systems*, 3(3):32–57, 1973.

[51] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[52] D. M. Eler, M. Y. Nakazaki, F. V. Paulovich, D. P. Santos, G. F. Andery, M. C. F. Oliveira, J. Batista-Neto, and R. Minghim. Visual analysis of image collections. *The Visual Computer*, 25(10):923–937, 2009.

[53] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6): 1216–1223, 2007.

[54] A. Endert, P. Fiaux, and C. North. Semantic interaction for visual text analytics. In *Proceedings of the 2012 SIGCHI Conference on Human Factors in Computing Systems*, pages 473–482, 2012.

[55] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications*, 14:153–158, 1969.

[56] O. Ersoy, C. Hurter, F. V. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2364–2373, 2011.

[57] R. Etemadpour, R. Motta, J. G. d. S. Paiva, R. Minghim, M. C. F. de Oliveira, and L. Linsen. Perception-based evaluation of projection methods for multidimensional data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 21(1):81–94, Jan 2015.

[58] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 24(2):163–174, May 1995.

[59] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, pages 178–178. IEEE, 2004.

[60] I. K. Fodor. A survey of dimension reduction techniques, 2002. e-reports-ext.llnl.gov/pdf/240921.pdf.

[61] FrontEndART. SourceMeter static source code analysis solution for java, c/c++, c#, python and rpg, 2016. http://www.sourcemeter.com.

[62] H. Gall, M. Jazayeri, and C. Riva. Application of information visualization to the analysis of software release history. In *Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*, pages 132–139. Springer Vienna, 1999.

[63] E. Gansner, Y. Hu, and S. Kobourov. Gmap: Visualizing graphs and clusters as maps. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pages 201 –208, march 2010.

[64] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *2011 IEEE Pacific Visualization Symposium (PacificVis)*, pages 187–194. IEEE, 2011.

[65] O. Gascuel and M. Steel. Neighbor-joining revealed. *Molecular Biology and Evolution*, 23(11):1997–2000, 2006.

[66] M. Gleicher. Explainers: Expert explorations with crafted projections. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2042–2051, 2013. ISSN 1077-2626.

[67] M. Goebel and L. Gruenwald. A survey of data mining and knowledge discovery tools. *ACM SIGKDD Explorations Newsletter*, 1(1), 1999.

[68] Google Guice. Java dependency injection framework, 2016. github.com/google/guice.

[69] A. D. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labeled leaves. *Journal of Classification*, 3:335–348, 1986.

[70] J. Gower, S. Lubbe, and N. Roux. *Understanding biplots*. Wiley, 2011.

[71] M. Greenacre. *Biplots in practice*. Fundacion BBVA, 2010.

[72] M. Griebel, T. Preusser, M. Rumpf, M. A. Schweitzer, and A. Telea. Flow field clustering via algebraic multigrid. In *Proceedings of the 2004 IEEE Visualization*, pages 35–42, 2004.

[73] L. Guo, W. Zuo, T. Peng, and B. K. Adhikari. Attribute-based edge bundling for visualizing social networks. *Physica A: Statistical Mechanics and its Applications*, 438:48–55, 2015.

[74] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[75] A. Hanjalic. ClonEvol: Visualizing software evolution with code clones. In *Proceedings of the First IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–4. IEEE Press, 2013.

[76] A. Hanjalic. ClonEvol: Visualizing software evolution with code clones. In *Proc. IEEE VISSOFT*, pages 1–4, 2013.

[77] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. In *Proceedings of the 10th International Symposium on Graph Drawing*, pages 207–219. Springer, 2002.

[78] J. Hartigan. Printer graphics for clustering. *Journal of Statistical Computation and Simulation*, 4(3):187–213, 1975.

[79] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Statistics. Springer, 2 edition, 2009.

[80] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[81] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 833–840, 2002.

[82] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[83] D. Holten and J. J. van Wijk. Visual comparison of hierarchically organized data. *Computer Graphics Forum*, 27(3):759–766, 2008.

[84] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[85] B. Hughes. Trees and ultrametric spaces: a categorical equivalence. *Advances in Mathematics*, 189:148–191, 2004.

[86] C. Hurter, O. Ersoy, and A. Telea. Graph bundling by kernel density estimation. *Computer Graphics Forum*, 31(3pt1):865–874, 2012.

[87] C. Hurter, O. Ersoy, and A. C. Telea. Graph bundling by kernel density estimation. *Computer Graphics Forum*, 31(3):865–874, 2012.

[88] C. Hurter, O. Ersoy, S. I. Fabrikant, T. R. Klein, and A. C. Telea. Bundled visualization of dynamic graph and trail data. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1141–1157, October 2013. ISSN 1941-0506.

[89] S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):249–261, 2009.

[90] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proceedings of the First IEEE Conference on Visualization*, pages 361–378, 1990.

[91] P. Isenberg, F. Heimerl, S. Koch, T. Isenberg, P. Xu, C. Stolper, M. Sedlmair, J. Chen, T. Möller, and J. Stasko. Visualization publication dataset. Dataset: http://vispubdata.org/ , 2015. URL http://vispubdata.org/. Published Jun. 2015.

[92] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[93] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[94] J. Johnson. Visualizing textual redundancy in legacy source. In *Proc. CAS-CON*, pages 32–38, 1994.

[95] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17:2563–2571, 2011.

[96] I. Jolliffe. *Principal Component Analysis*. Springer, 3 edition, 2002.

[97] JUnit. JUnit testing framework, 2016. github.com/junit-team/junit4.

[98] T. Kamiya. CCfinderX clone detector, 2014. http://www.ccfinder.net.

[99] E. Kandogan. Just-in-time annotation of clusters, outliers, and trends in point-based data visualizations. In *IEEE Symposium on Visual Analytics Science and Technology (VAST 2012)*, pages 73–82, Oct 2012.

[100] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering the information age: Solving problems with visual analytics (VisMaster)*. Eurographics Association, 2010.

[101] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[102] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, Sep 1990. ISSN 0018-9219.

[103] Y. Koren, L. Carmel, and D. Harel. Drawing huge graphs by algebraic multigrid optimization. *Multiscale Modeling & Simulation*, 1(4):645–673, 2003.

[104] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[105] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. ISSN 1860-0980.

[106] A. Kuhn, D. Erni, P. Loretan, and O. Nierstrasz. Software cartography: Thematic software visualization with consistent layout. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(3):191–210, Apr. 2010.

[107] L. I. Kuncheva. *Fuzzy Classifier Design*. Physica-Verlag, 2010.

[108] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862, 2010.

[109] C. Lange, M. Wijns, and M. Chaudron. MetricViewEvolution: UML-based views for monitoring model evolution and quality. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, pages 327–328. IEEE Press, 2007.

[110] M. Lanza. The evolution matrix: recovering software evolution using software visualization techniques. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 37–42. ACM, 2001.

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

[112] P. A. Laplante. *What Every Engineer Should Know About Software Engineering*. CRC Press, Inc., Boca Raton, FL, USA, 2007. ISBN 0849372283.

[113] J. Li and J. Z. Wang. Automatic linguistic indexing of pictures by a statistical modelin approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1075–1088, 2003.

[114] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. In *Proceedings of the Eurographics Conference on Visualizaiton (EuroVis 2015) – STARs*, 2015.

[115] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40:262–282, 2007.

[116] S. G. MacDonell. Visualization and analysis of software engineering data using self-organizing maps. In *Proceedings of the International Symposium on Empirical Software Engineering*, pages 233–242, 2005.

[117] C. Marinescu, R. Marinescu, P. F. Mihancea, and R. Wettel. iplasma: An integrated platform for quality assessment of object-oriented design. In *Proceedings of the 2005 International Conference on Software Maintenance*, pages 77–80. IEEE Computer Society Press, 2005.

[118] R. Martins, D. Coimbra, R. Minghim, and A. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.

[119] R. M. Martins. *Explanatory Visualization of Multidimensional Projections*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Groningen, the Netherlands, March 2016.

[120] R. M. Martins, R. Minghim, and A. C. Telea. Explaining neighborhood preservation for multidimensional projections. In R. Borgo and C. Turkay,

editors, *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association, 2015.

[121] P. Meirelles, C. Santos, J. Miranda, F. Kon, A. Terceiro, and C. Chavez. A study of the relationships between source code metrics and attractiveness in free software projects. In *Proceedings of the 2010 Brazilian Symposium on Software Engineering (SBES)*, pages 11–20, Sept 2010. doi: 10.1109/SBES.2010.27.

[122] T. Mens and S. Demeyer. *Software Evolution*. Springer, 2008.

[123] I. Mierswa and K. Morik. Automatic feature extraction for classifying audio data. *Machine Learning*, 58(2-3):127–149, 2005.

[124] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1 edition, 1997.

[125] D. Moura. 3D density histograms for criteria-driven edge bundling, 2015. *arXiv:1504.02687v1* [cs.GR].

[126] T. Munzner. *Visualization Analysis and Design*. CRC Press, 2015.

[127] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, June 2006. ISSN 0027-8424.

[128] G. Nguyen and M. Worring. Interactive access to large image collections using similarity-based visualization. *Journal of Visual Languages & Computing*, 19(2):203–224, 2008.

[129] Q. Nguyen, P. Eades, and S.-H. Hong. Streameb: Stream edge bundling. In *Graph Drawing*, pages 400–413. Springer, 2013.

[130] M. Nixon. *Feature Extraction & Image Processing for Computer Vision*. Academic Press, 3 edition, 2012.

[131] A. Nocaj and U. Brandes. Organizing search results with a reference map. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2546–2555, 2012.

[132] A. Nocaj and U. Brandes. Computing voronoi treemaps: Faster, simpler, and resolution-independent. *Computer Graphics Forum*, 31(3pt1):855–864, 2012. ISSN 1467-8659.

[133] U. of Maryland. US counties dataset, 2014. URL http://archive.ics.uci.edu/ml.

[134] G. Oliveira, J. Comba, R. Torchelsen, M. Padilha, and C. Silva. Visualizing running races through the multivariate time-series of multiple runners. In *Proceedings of the XXVI Conference on Graphics, Patterns and Images (SIBGRAPI 2013)*, pages 99–106, 2013.

[135] P. Pagliosa, F. Paulovich, R. Minghim, H. Levkowitz, and L. Nonato. Projection inspector: Assessment and synthesis of multidimensional projections. *Neurocomputing*, 150:599–610, 2015.

[136] J. G. Paiva, L. Florian, H. Pedrini, G. Telles, and R. Minghim. Improved similarity trees and their application to visual data classification. *IEEE Transactions on Visualization and Computer Graphics*, 17:2459–2468, 2011.

[137] N. R. Pal and J. C. Bezdek. On cluster validity for the fuzzy c-means model. *IEEE Transactions on Fuzzy Systems*, 3(3):370–379, 1995.

[138] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *Proceedings of the Fourth ACM International Conference on Multimedia*, pages 65–73, New York, NY, USA, 1996.

[139] F. Paulovich, D. Eler, J. Poco, C. Botha, R. Minghim, and L. Nonato. Piecewise laplacian-based projection for interactive data exploration and organization. *Computer Graphics Forum*, 30(3):1091–1100, 2011.

[140] F. V. Paulovich and R. Minghim. Hipp: A novel hierarchical point placement strategy and its application to the exploration of document collections. *IEEE Transactions on Visualization and Computer Graphics*, 14(6): 1229–1236, 2008. ISSN 1077-2626.

[141] 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 Transactions on Visualization and Computer Graphics*, 14(3):564–575, 2008.

[142] F. V. Paulovich, C. T. Silva, and L. G. Nonato. Two-phase mapping for projecting massive data sets. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1281–1290, 2010.

[143] F. V. Paulovich, G. P. Telles, F. M. B. Toledo, R. Minghim, and L. G. Nonato. Semantic wordification of document collections. *Computer Graphics Forum*, 31:1145–1153, 2012.

[144] G. A. Pavlopoulos, T. G. Soldatos, A. Barbosa-Silva, and R. Schneider. A reference guide for tree analysis and visualization. *BioData Mining*, 3:1, 2010.

[145] W. Pedrycz, G. Succi, M. Reformat, P. Musilek, and X. Bai. Self organizing maps as a tool for software analysis. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 93–97 vol.1, 2001.

[146] E. Pekalska, D. de Ridder, R. Duin, and M. Kraaijveld. A new method of generalizing Sammon mapping with application to algorithm speed-up. In

Proceedings of the 5th Annual Conference of the Advanced School for Computing and Imaging (ASCI), pages 221–228, Delft, NL, 1999.

[147] V. Peysakhovich, C. Hurter, and A. Telea. Attribute-driven edge bundling for general graphs with applications in trail analysis. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2015.

[148] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, INFOVIS '05, Washington, DC, USA, 2005. IEEE Computer Society.

[149] C. Plaisant, J. Grosjean, and B. B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2002)*, pages 57–64. IEEE, 2002.

[150] R. Rao and S. Card. The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 318–322, 1994.

[151] P. Rauber, A. Falcao, and A. Telea. Visualizing time-dependent data using dynamic t-SNE. In *Proceeding of the 2015 Eurographics Conference on Visualization (EuroVis 2015) – Short papers*. Eurographics Association, 2016.

[152] P. E. Rauber, R. R. O. da Silva, S. Feringa, M. E. Celebi, A. Falcão, and A. Telea. Interactive Image Feature Selection Aided by Dimensionality Reduction. In *Proceedings of the 2015 EuroVis Workshop on Visual Analytics (EuroVA 2015)*, pages 54–61, 2015.

[153] M. Reformat, W. Pedrycz, and N. J. Pizzi. Software quality analysis with the use of computational intelligence. In *Proceedings of the 2002 IEEE International Conference on Fuzzy Systems*, volume 2, pages 1156–1161, 2002.

[154] D. Reniers, L. Voinea, O. Ersoy, and A. Telea. The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product. *Science of Computer Programming*, 79:224–240, 2014.

[155] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[156] C. Roy, J. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, 2008.

[157] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4): 406–425, 1987.

[158] G. Salton. Developments in automatic text retrieval. *Science*, 253(5023): 974, 1991.

[159] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

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

[161] H. J. Schulz. Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[162] H. J. Schulz, S. Hadlak, and H. Schumann. Point-based tree representation: A new approach for large hierarchies. In *2009 IEEE Pacific Visualization Symposium (PacificVis)*, pages 81 –88, 2009.

[163] SciTools. Understand static code analysis tool, 2016. `https://scitools.com`.

[164] M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2634–2643, 2013.

[165] D. Selassie, B. Heller, and J. Heer. Divided edge bundling for directional network data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2354–2363, 2011.

[166] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22(1–3):21–74, 2002. `https://www.cs.cmu.edu/~quake/triangle.html`.

[167] F. Sikansi, R. R. O. da Silva, R. Minghim, and F. V. Paulovich. Similarity-driven Edge Bundling: Revisiting Hierarchical Edge Bundling for semantic meaningful clutter reduction in graphs layouts. Manuscript in preparation.

[168] V. Silva and J. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford University, 2004.

[169] M. Simonsen, T. Mailund, and C. N. Pedersen. Rapid neighbour-joining. In *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics (WABI 2008)*, pages 113–122, Karlsruhe, Germany, 2008.

[170] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[171] C. Sorzano, J. Vargas, and A. Montano. A survey of dimensionality reduction techniques, 2014.

[172] R. Spence. *Information Visualization – An Introduction*. Springer, 2014.

[173] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 151–162, London, UK, UK, 2001. Springer-Verlag. ISBN 3-211-83709-4.

[174] R. O. Stehling, M. A. Nascimento, and A. X. Falcão. A compact and efficient image retrieval approach based on border/interior pixel classification. In *Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM 2002)*, CIKM '02, pages 102–109, 2002.

[175] R. J. Sternberg. *Cognitive Psychology*. Harcourt Brace College Publishers, 2 edition, 1996.

[176] E. A. Stohr and B. R. Konsynski. *Information Systems and Decision Processes*. IEEE Computer Society Press, 1992.

[177] M. Strickert, S. Teichmann, N. Sreenivasulu, and U. Seiffert. High-throughput multi-dimensional scaling (hit-mds) for cDNA-array expression data. In *Proceedings of the 15th International Conference Artificial Neural Networks*, volume 3696, pages 625–633. Springer, 2005.

[178] M. Sun, P. Mi, C. North, and N. Ramakrishnan. BiSet: Semantic edge bundling with biclusters for sensemaking. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):310–319, 2016.

[179] Tableau, Inc. Tableau visualization framework, 2016. http://www.tableau.com.

[180] L. Tan, Y. Song, S. Liu, and L. Xie. Imagehive: Interactive content-aware image summarization. *IEEE Computer Graphics and Applications*, 32(1):46–55, 2012.

[181] S. Taylor and M. Alexander. Science, technology, and human factors in fire danger rating: the Canadian experience. *International Journal of Wildland Fire*, 15(15):121–135, 2006.

[182] A. Telea. Combining extended table lens and treemap techniques for visualizing tabular data. In *Proceeding of the 2007 Eurographics Conference on Visualization (EuroVis 2007)*, pages 51–58, 2007.

[183] A. Telea. *Data visualization – principles and practice*. CRC Press, 2 edition, 2014.

[184] A. Telea and D. Auber. Code flows: Visualizing structural evolution of source code. *Computer Graphics Forum*, 27(3):831–838, 2008.

[185] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Computer Graphics Forum*, 29(3):843–852, 2010.

[186] A. Telea, A. Maccari, and C. Riva. An open toolkit for prototyping reverse engineering visualizations. In *Proc. VisSym*, pages 241–249. Springer, 2002.

[187] A. C. Telea. *Data Visualization: Principles and Practice, Second Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2014. ISBN 1466585269, 9781466585263.

[188] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

[189] M. Termeer, C. Lange, A. Telea, and M. Chaudron. Visual exploration of combined architectural and metric information. In *Proceedings of the* 3rd *IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 21–26, 2005.

[190] E. Tufte. *Beatiful evidence*. Graphics Press, 2006.

[191] E. R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, CT, 1990.

[192] C. Turkay, P. Filzmoser, and H. Hauser. Brushing dimensions – a dual visual analysis model for high-dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2591–2599, 2011.

[193] L. van der Maaten and G. E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[194] M. van der Zwan, V. Codreanu, and A. Telea. CUBu: Universal real-time bundling for large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 2016.

[195] R. van Liere and W. de Leeuw. Graphsplatting: Visualizing graphs as continuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 9 (2):206–212, 2003.

[196] J. J. Van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proc. IEEE InfoVis*, pages 73–82, 1999.

[197] K. Vehkalahti. Biplots in practice by michael greenacre. *International Statistical Review*, 79(1):136–137, 2011. ISSN 1751-5823.

[198] R. Vliegen, J. J. V. Wijk, and E. jan Van Der Linden. Visualizing business data with generalized treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):789 – 796, 2006.

[199] L. Voinea and A. Telea. CVSgrab: Mining the history of large software projects. In *Proceedings of the 2006 Eurographics / IEEE VGTC Symposium on Visualization*, pages 187–194. IEEE Press, 2006.

[200] L. Voinea and A. Telea. Visual clone analysis with SolidSDD. In *Proc. IEEE VISSOFT*, pages 87–94, 2014.

[201] L. Voinea, A. Telea, and J. J. van Wijk. Cvsscan: Visualization of code evolution. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, pages 47–56, 2005.

[202] 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. *Computer Graphics Forum*, 30(6): 1719–1749, 2011.

[203] C. Wang, J. P. Reese, H. Zhang, J. Tao, Y. Gu, J. Ma, and R. J. Nemiroff. Similarity-based visualization of large image collections. *Information Visualization*, 2013.

[204] J. T.-L. Wang, X. Wang, K.-I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD, pages 307–311, New York, NY, USA, 1999. ACM. ISBN 1-58113-143-7.

[205] R. Wettel and M. Lanza. Program comprehension through software habitability. In *Proceedings of the 15th IEEE International Conference on Program Comprehension*, 2007.

[206] T. J. Wheeler. Large-scale neighbor-joining with ninja. In *Proceedings of the 9th International Workshop on Algorithms in Bioinformatics (WABI 2009)*, pages 375–389, Philadelphia, PA, USA, 2009.

[207] P. C. Wong and J. Thomas. Visual analytics. *IEEE Computer Graphics and Applications*, 24(5):20–21, 2004.

[208] C. Xu and J. Prince. Gradient vector flow: A new external force for snakes. In *Proceedings of the 1997 IEEE Conference on Computer Vision and Pattern Recognition*, pages 66–71, 1997.

[209] T. Yamashita and R. Saga. Edge bundling in multi-attributed graphs. In S. Yamamoto, editor, *Human Interface and the Management of Information. Information and Knowledge Design*, volume 9172 of *Lecture Notes in*

Computer Science, pages 138–147. Springer International Publishing, 2015. ISBN 978-3-319-20611-0.

[210] I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.

[211] J. S. Yi, R. Melton, J. Stasko, and J. A. Jacko. Dust & magnet: multivariate information visualization using a magnet metaphor. *Information Visualization*, 4(4):239–256, 2005.

[212] A. T. T. Ying. Mining challenge 2015: Comparing and combining different information sources on the stack overflow data set. In *The 12th Working Conference on Mining Software Repositories*, page to appear, 2015.

[213] H. Zhou, P. Xu, X. Yuan, and H. Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, 2013.

List of Figures

List of Publications

The following publications resulted from the work presented in this thesis:

- R. R. O. da Silva, P. Rauber, R. M. Martins, R. Minghim, and A. Telea. Attribute-based visual explanation of multidimensional projections. In *Proceedings of the 2015 EuroVis Workshop on Visual Analytics (EuroVA 2015)*, pages 137–142, 2015

- R. R. O. da Silva, E. F. Vernier, P. E. Rauber, J. L. D. Comba, R. Minghim, and A. Telea. Metric Evolution Maps: Multidimensional attribute-driven exploration of software repositories. Accepted for publication in the 21st International Symposium on Vision, Modeling and Visualization (VMV)., 2016

- R. R. O. da Silva, J. G. de Souza Paiva, G. P. Telles, F. Rolli, C. E. A. Zampieri, and R. Minghim. The Visual Supertree: Similarity-based multiscale visualization. Submitted to Information Visualization (IVI), 2016

- F. Sikansi, R. R. O. da Silva, R. Minghim, and F. V. Paulovich. Similarity-driven Edge Bundling: Revisiting Hierarchical Edge Bundling for semantic meaningful clutter reduction in graphs layouts. Manuscript in preparation

Other publications during the development of this thesis:

- P. E. Rauber, R. R. O. da Silva, S. Feringa, M. E. Celebi, A. Falcão, and A. Telea. Interactive Image Feature Selection Aided by Dimensionality Reduction. In *Proceedings of the 2015 EuroVis Workshop on Visual Analytics (EuroVA 2015)*, pages 54–61, 2015

- R. R. O. da Silva, P. E. Rauber, and A. C. Telea. Beyond the third dimension: Visualizing high-dimensional data with projections. *IEEE Computing in Science & Engineering*, 2016. Accepted for publication

Acknowledgments

To my family: For supporting me in learning, pursuing my dreams and always being on my side. I will always be grateful to you.

To my beloved Gleice: For your endless support, love and companion. I am very lucky to have you in my life, and I dearly hope to enjoy your companionship for many years to come.

To Alex: For being a huge example of professional, leader and supervisor. I am really thankful for the opportunity to work with you and have all the nice discussions, from which I have learned a lot. Thank you.

To Rosane: For being another huge example of supervisor. You have introduced me to the journey into the field of research, and always believed in me. I will always be grateful for your support, guidance and patience.

To my Groningen friends: For being part of my life, making me feel at home, and sharing great moments together.

To all SVCG and VICG colleagues: For all nice discussions and invaluable help during the period of this PhD. Certainly this work has many of your contributions. Thank you.

To the assessment committee: For dedicating your time to read my thesis and give me great improvement suggestions.

To the Flaticon designers: I would like to thank the graphics designers Gregor Cresnar, Freepik and Zlatko Najdenovski from Flaticon (www.flaticon.com), for designing such great set of icons and providing it for download in their webpage.

Colophon

This thesis was typeset with $\LaTeX 2_{\varepsilon}$ using Robert Slimbach's *Minion Pro* font. The typesetting is based on the *classicthesis* style by André Miede.