Efficient Point Clustering for Visualization

Dissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
vorgelegt von

M.Sc. Computer Science
Christian Joachim Beilschmidt
geboren in Siegburg

Marburg, im Februar 2019

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg (Hochschulkennziffer 1180) als Dissertation am 4. Juli 2019 angenommen.

Erstgutachter: Prof. Dr. Bernhard Seeger, Philipps-Universität Marburg

Zweitgutachter: Prof. Dr. Thorsten Thormählen, Philipps-Universität Marburg

Tag der Einreichung: 25. Februar 2019

Tag der mündlichen Prüfung: 8. August 2019

—Dedicated to my one decade of computer science studies

Abstract

The visualization of large spatial point data sets constitutes a problem with respect to runtime and quality. A visualization of raw data often leads to occlusion and clutter and thus a loss of information. Furthermore, particularly mobile devices have problems in displaying millions of data items. Often, thinning via sampling is not the optimal choice because users want to see distributional patterns, cardinalities and outliers. In particular for visual analytics, an aggregation of this type of data is very valuable for providing an interactive user experience. This thesis defines the problem of visual point clustering that leads to proportional circle maps. It furthermore introduces a set of quality measures that assess different aspects of resulting circle representations.

The Circle Merging Quadtree constitutes a novel and efficient method to produce visual point clusterings via aggregation. It is able to outperform comparable methods in terms of runtime and also by evaluating it with the aforementioned quality measures. Moreover, the introduction of a preprocessing step leads to further substantial performance improvements and a guaranteed stability of the Circle Merging Quadtree. This thesis furthermore addresses the incorporation of miscellaneous attributes into the aggregation. It discusses means to provide statistical values for numerical and textual attributes that are suitable for side-views such as plots and data tables. The incorporation of multiple data sets or data sets that contain class attributes poses another problem for aggregation and visualization. This thesis provides methods for extending the Circle Merging Quadtree to output pie chart maps or maps that contain circle packings. For the latter variant, this thesis provides results of a user study that investigates the methods and the introduced quality criteria.

In the context of providing methods for interactive data visualization, this thesis finally presents the VAT System, where VAT stands for visualization, analysis and transformation. This system constitutes an exploratory geographical information system that implements principles of visual analytics for working with spatio-temporal data. This thesis details on the user interface concept for facilitating exploratory analysis and provides the results of two user studies that assess the approach.

Zusammenfassung

Die Visualisierung großer räumlicher Punktdatensätze stellt ein Problem in Bezug auf Laufzeit und Qualität dar. Eine Visualisierung von Rohdaten führt oft zu Verdeckungen und Unübersichtlichkeiten und damit zu einem Verlust von Informationen. Darüber hinaus haben insbesondere mobile Geräte Probleme bei der Darstellung von Millionen von Datenelementen. Oft ist die Ausdünnung durch Sampling nicht die optimale Wahl, da die Anwender Verteilungsmuster, Kardinalitäten und Ausreißer erkennen wollen. Insbesondere für die Visual Analytics ist eine Aggregation dieser Art von Daten sehr wertvoll, um eine interaktive Benutzerführung zu ermöglichen. Diese Dissertation definiert das Problem des Visual Point Clusterings, das zu proportionalen Kreiskarten führt. Darüber hinaus wird eine Reihe von Qualitätsmaßnahmen eingeführt, die verschiedene Aspekte einer resultierenden Kreisdarstellung bewerten.

Der Circle Merging Quadtree stellt eine neuartige und effiziente Methode zur Erzeugung von Visual Point Clusterings durch Aggregation dar. Er ist in der Lage, vergleichbare Methoden in Bezug auf die Laufzeit zu schlagen und sie auch mit den oben genannten Qualitätsmaßnahmen zu validieren. Darüber hinaus führt die Einführung eines Preprocessing-Schrittes zu weiteren deutlichen Leistungssteigerungen und einer garantierten Stabilität des Circle Merging Quadtrees. Diese Arbeit befasst sich darüber hinaus mit der Einbeziehung weiterer Attribute in die Aggregation. Es werden Mittel zur Berechnung statistischer Werte für numerische und textuelle Attribute diskutiert, die für Seitenansichten wie Diagramme und Datentabellen geeignet sind. Die Einbindung mehrerer Datensätze oder Datensätze, die Klassenattribute enthalten, stellt ein weiteres Problem für die Aggregation und Visualisierung dar. Diese Arbeit stellt Methoden für die Erweiterung des Circle Merging Quadtrees zur Verfügung, um Pie-Chart-Karten oder Karten, die Circle Packings enthalten, auszugeben. Für die letztgenannte Variante liefert diese Arbeit Ergebnisse einer Nutzerstudie, die die Methoden und eingeführten Qualitätsmaße untersucht.

Im Rahmen der Bereitstellung von Methoden für die interaktive Datenvisualisierung stellt diese Dissertation schließlich das VAT System vor, wobei VAT für Visualisierung, Analyse und Transformation steht. Dieses System stellt ein explorativ geografisches Informationssystem dar, das die Prinzipien der visuellen Analytik für die Arbeit mit raumzeitlichen Daten umsetzt. Diese Arbeit beschreibt das Konzept der Benutzeroberfläche zur Unterstützung der explorativen Analyse und liefert dabei die Ergebnisse von zwei Benutzerstudien, die den Ansatz bewerten.

Erklärung

Hiermit versichere ich, dass ich meine Dissertation mit dem Titel

Efficient Point Clustering for Visualization

selbständig und ohne fremde Hilfe verfasst, nicht andere als die in ihr angegebenen Quellen oder Hilfsmittel benutzt, alle vollständig oder sinngemäß übernommenen Zitate als solche gekennzeichnet sowie die Dissertation in der vorliegenden oder einer ähnlichen Form noch bei keiner anderen in- oder ausländischen Hochschule anlässlich eines Promotionsgesuchs oder zu anderen Prüfungszwecken eingereicht habe. Dies ist mein erster Versuch einer Promotion.

Marburg, den 25. Februar 2019 Christian Beilschmidt

Acknowledgments

I foremost want to express my gratitude towards Prof. Dr. Bernhard Seeger for supervising me through the course of this thesis and for bringing me into his group. All discussions with him were very valuable and always encouraging to conduct my research that lead to this thesis.

I want to thank all my colleagues from the Database Research Group of the University of Marburg for their collaboration in the course of my doctoral studies: Dr. Daniar Achakeev, Johannes Drönner, Nikolaus Glombiewski, Jana Holznigenkemper, Dr. Bastian Hoßbach, Michael Körber, Michael Mattig, Andreas Morgen and Marc Seidemann. Here, I would like to particularly mention Michael Mattig and Johannes Drönner. Michael supported me in all my research and was the best possible co-author I could imagine. Further thanks for the time he spent proofreading this thesis. Johannes provided me with all his expertise in the field of geography and, together with Michael, he did an excellent teamwork in the development of the Vat System. With respect to the Vat System I like to thank the students Sören Hoffstedt, Julian Märte, Bastian Reitemeier, Daniel Schneider and Kerstin Winter for spending particular implementation effort in advancing the system. I furthermore want to thank Mechthild Keßler for supporting me in all organizational aspects, in particular with respect to contract renewals.

For support from outside the Database Research Group I want to particularly thank Dr. Thomas Fober for participating in my research and advising me during his time as Interim Professor at the University of Marburg. Moreover, I enjoyed working together on data science lectures. Furthermore, I want to thank Dr. Alexander Markowetz in my time as a student at the University of Bonn for his recommendation for a doctorate.

I would like to thank my family and friends who have supported me during this time, although I was often away from the Cologne-Bonn area. In particular, I would like to thank Verena for all her support and for proofreading this thesis, which hopefully improved my English proficiency.

My work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under grant no. SE 553/7-1 and SE 553/7-2. As part of the GFBio project I am particularly thankful for the provision of use cases of biodiversity research that lead to the idea of visual point clustering and to the development of the Visualization, Analysis and Transformation System.

Contents

1 Introduction

In recent years, data-driven research has become almost ubiquitous in science. A key aspect is the exploratory investigation of available data, which leads to new ideas and hypotheses and which again leads to in-depth analyses. For instance, Andrienko et al. [And+10] describe the important task of analyzing the environment and the effects of climate change through visual analytics. In the context of visual analytics, a discipline where cognitive strengths of humans are combined with powerful tooling from computers, data visualization is of utmost importance. The rapidly growing complexity and size of data sets make this visualization task especially challenging. Particularly the visualization of spatial data, which is present in applications like transportation and business analytics, can benefit from new visualization technology [Kei+08b]. Kraak [Kra04] states that spatial data is of special interest because the visualization of data on a map particularly stimulates recognizing spatial patterns and relationships.

While visualization has been primarily performed on dedicated hardware so far, there is currently a tendency that mobile devices are becoming more popular as a front end for the analysis, and thus, replace the analytical usage of desktops [MR05]. Thus, tablets and smartphones are increasingly becoming the target devices for developers. Data visualization must therefore adapt to the limitations of Internet bandwidth, computing power, battery capacity and screen size. On the other hand, the era of big data leads to scenarios in which analyses are required to process huge amounts of data [Zha+12; Mad+12]. Hence, it is necessary to visualize more and more data on a constrained screen size and on devices that have limited processing capabilities. A suitable way to address this challenge is to aggregate data on the server side and to only transfer the aggregated data to the mobile devices. This approach enables interactive analysis, while it is still efficient in bandwidth and battery power on mobile devices.

Aside from computational aspects, visualizing raw data has additional downsides. The limited screen size generally means having constrained maps that suffer from a highly overloaded and occluded view. In this case, data aggregation can also help

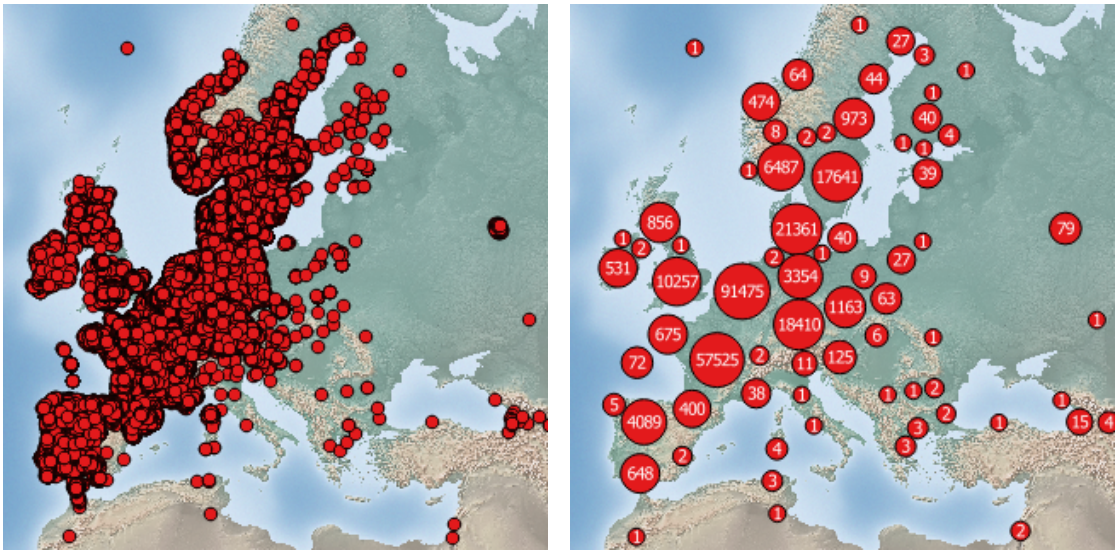


Figure 1.1.: This figure shows the discrepancy between a plot of raw points (left) of the black alder and an aggregated view (right) of the points [Bei+17d].

to achieve a clear visualization with a higher information content. In particular for detecting patterns like hot spots, an appropriate degree of aggregation facilitates to focus on important aspects. Moreover, also the detection of outliers is a crucial task that facilitates finding either errors or interesting phenomena. Figure 1.1 emphasizes this problem by showing raw data of the black alder on a map on the left side. There are data points in western, central and northern Europe. It clearly indicates the importance of data aggregation because most of the data is completely occluded. The right side shows an aggregated circle map that resolves this problem. For example, due to the size of the circles and the indicated number of points, it is obvious that most occurrences are in central Europe, more precisely in France and the Benelux Union. In contrast, the raw visualization does not provide the information that France contains more than an order of magnitude more points than, for instance, Spain or Norway. In the following sections, we explain our interest in the biodiversity domain and first present a real project in which data visualization is a key element. Then, we briefly introduce the setting of biodiversity data.

1.1. GFBio

The German Federation for Biological Data (GFBio) is a German infrastructure project that aims to integrate biodiversity data of German research collections,

data centers and archives [DGG+14]. On the one hand, the integration includes the digital and standardized provision of data from, for instance, collections of museums. On the other hand, it also involves offering a unified access point for planning the data management of (new) research projects and finding the most competent and suitable corresponding GFBio partner. Many research data is not fit for reuse because of missing data standards and strategies for long-term storage. GFBio defines common standard workflows and data formats that address this problem.

In addition to facilitating data reuse, the GFBio project also develops added-value services for accessing, visualizing and processing available data. As a member of the database research group of the University of Marburg we are developing the Vat system as part of GFBio. This system offers a powerful visualization and processing engine to integrate and combine research data from GFBio on the one hand, but also other public resources on the other hand. It facilitates visual analytics principles and supports users in fulfilling their research tasks. Furthermore, the system integrates into the GFBio portal, which is a set of services that are interconnected. In particular, the data search [Löf+17] allows users to look up data using a unified search index that combines all data sources within GFBio.

1.2. Occurrence and Biodiversity Data

Throughout this thesis, we mainly use data examples from the biodiversity domain, in more detail occurrence data. Occurrence data constitutes points that indicate the location where species were observed or collected geographically. For instance, researchers observe European wildcats as part of their studies. They collect these points via a GPS device and also note down miscellaneous attributes like the time of observation and the fur color. In most cases it is not guaranteed that observations in a data set distinguish unique animals, e.g. one animal could occur several times and others only once just by chance. Furthermore, the absence of occurrence records at a certain place does not imply that there actually is no species apparent, i.e., there are generally no negative findings recorded and the data is not complete.

The data quality of occurrence data sets is very heterogeneous. There are data sets generated by amateur researchers as well as renowned experts. Issues are, for example, the confusion of species with another, incorrect or flipped GPS coordinates,

inaccurate old data and problems that arise during data import. Another common error is the occurrence of records with coordinate $(0, 0)$ that was mistakenly introduced as a replacement of *unknown* or *null* values.

Apart from data quality, this type of data is interesting for analytical algorithms because of its inherent heterogeneity. Some data sets have rather few data points and some data sets are very large. Furthermore, the distribution of species varies significantly. Due to geographical factors, but also because of invasion, there are species that only occur very locally in small regions or very widely on entire continents or worldwide. In this scenario, hot spots of species are very important, but outliers are also of interest. Outliers can indicate either errors, which are relevant for data curators, or interesting findings, which are relevant for researchers.

The Global Biodiversity Information Facility[1] (GBIF) is the largest collection of occurrence data of living species. The organization was established in 2001 and has since been supported by the science ministers of the OECD. It provides open access to its data. As of December 2018, 1 315 worldwide institutions have published over 40 000 data sets and contributed more than one billion occurrence records. This includes data from one to two million species in Darwin Core standard [Wie+12]. This standard is endorsed by the organization for Biodiversity Information Standards[2] (TDWG). It includes, among other fields, species names that follow the GBIF taxonomy as well as a location and time information.

1.3. Contributions

The research contributions of this thesis are listed in the following:

- Various algorithms are candidates for solving the visual point clustering problem. However, it is difficult to assess the quality of a result in a formal way rather than by surveys with domain experts. This thesis provides a comprehensive set of quality measures for visual point clustering. Furthermore, it presents a survey of state-of-the-art point aggregation methods with focus on clustering and experiments with different domain data sets to inspect the validity of the quality measures.

[1] www.gbif.org
[2] www.tdwg.org

- Visualizing large spatial data sets in their raw form often leads to inadequate results that suffer from occlusion, and thus a loss of information. Moreover, technical constraints, particularly for mobile devices, do not allow displaying millions of data points. This thesis defines the problem of visual point clustering and introduces the Circle Merging Quadtree, a novel algorithm for ad-hoc aggregation and visualization of large point data sets that is suitable for big data scenarios. The resulting proportional circle maps provide the user with important information such as hot spots, distributional patterns and outliers.
- The Circle Merging Quadtree constitutes an efficient method for point aggregation, but does not guarantee stability. This thesis presents a preprocessing step for this algorithm that leads to stable results with respect to input permutations. Moreover, this preprocessing step leads to substantial performance improvements for the Circle Merging Quadtree.
- This thesis contains an extensive evaluation that demonstrates the superiority of the Circle Merging Quadtree algorithm over comparable algorithms with respect to runtime and quality.
- Spatial data often has miscellaneous attributes that are of interest to researchers as well. The Circle Merging Quadtree aggregates data sets using their spatial properties. This thesis presents an overview of aggregation methods that are suitable for providing a side-view, e.g. a plot or a data table, with aggregates of the miscellaneous attributes.
- The visualization of multiple data sets poses similar problems to the visualization of raw data like occlusion and many overlaps. This thesis presents the adaptation of two approaches from literature to the Circle Merging Quadtree, pie chart maps and maps of circle packings. In particular for the latter variant, this thesis presents extensions to known approaches and provides a user study that investigates the effect of this map type and its extensions.
- The Visualization, Analysis and Transformation System (Vat) constitutes an interactive geographical information system for the exploratory analysis of spatio-temporal data that applies techniques from visual analytics. This thesis presents an interface for exploratory workflow creation, effective data generalization and previews, linked time series computations, and automatic provenance and citation tracking. For data generalization, it uses the Circle Merging Quadtree as main method for visualizing point data sets. Additionally, this paper presents the results of two user studies that verify the validity of the approach.

The following papers were published in the course of this thesis:

- Christian Beilschmidt, Michael Mattig, Thomas Fober, Bernhard Seeger: **An Efficient Aggregation and Overlap Removal Algorithm for Circle Maps.**
  GeoInformatica (2019) 23: 473.
- Christian Beilschmidt, Thomas Fober, Michael Mattig, Bernhard Seeger: **A Linear-Time Algorithm for the Aggregation and Visualization of Big Spatial Point Data.**
  SIGSPATIAL/GIS 2017: 73:1-73:4.
- Christian Beilschmidt, Thomas Fober, Michael Mattig, Bernhard Seeger: **Quality Measures for Visual Point Clustering in Geospatial Mapping.**
  W2GIS 2017: 153-168.
- Christian Beilschmidt, Johannes Drönner, Michael Mattig, Marco Schmidt, Christian Authmann, Aidin Niamir, Thomas Hickler, Bernhard Seeger: **VAT: A Scientific Toolbox for Interactive Geodata Exploration.**
  Datenbank-Spektrum 17(3): 233-243 (2017).
- Christian Beilschmidt, Johannes Drönner, Michael Mattig, Bernhard Seeger: **VAT: A System for Data-Driven Biodiversity Research.**
  EDBT 2017: 546-549.
- Christian Beilschmidt, Johannes Drönner, Michael Mattig, Marco Schmidt, Christian Authmann, Aidin Niamir, Thomas Hickler, Bernhard Seeger: **Interactive Data Exploration for Geoscience.**
  BTW Workshops 2017: 117-126.
- Christian Authmann, Christian Beilschmidt, Johannes Drönner, Michael Mattig, Bernhard Seeger: **VAT: A System for Visualizing, Analyzing and Transforming Spatial Data in Science.**
  Datenbank-Spektrum 15(3): 175-184 (2015).
- Christian Authmann, Christian Beilschmidt, Johannes Drönner, Michael Mattig, Bernhard Seeger:
  Rethinking Spatial Processing in Data-Intensive Science.
  BTW Workshops 2015: 161-170.

1.4. Organization of the Thesis

The rest of this thesis is structured as follows:

Chapter 2 presents and defines the problem of visual point clustering. It introduces fundamental concepts and discusses assumptions.

This chapter contains parts of [Bei+17d].

Chapter 3 presents fundamentals as well as related work in the context of the thesis. It outlines the background of geographical information systems and visual analytics and introduces spatial data structures and algorithms. Moreover, it provides an overview of clustering in general and also discusses generic clustering quality measures.

This chapter contains parts of [Bei+17e], [Bei+17d] and [Bei+19].

Chapter 4 introduces quality measures for results of visual point clustering algorithms and provides detailed definitions. In this sense, it shows an evaluation of existing clustering algorithms and outlines their strengths and weaknesses.

This chapter contains parts of [Bei+17e].

Chapter 5 presents the Circle Merging Quadtree as an efficient algorithm for visual point clustering. It describes a preprocessing step that leads to substantial performance improvements and ensures stability of the Circle Merging Quadtree. Furthermore, it shows results of experimental evaluations with respect to runtime, quality, stability and compression.

This chapter contains parts of [Bei+17d] and [Bei+19].

Chapter 6 provides extensions with respect to the incorporation of miscellaneous attributes and multiple data sets or classes in the Circle Merging Quadtree method. The first section deals with the aggregation of these miscellaneous attributes for side-views. The second section presents two methods for visualizing multiple data sets or multiple classes in the data alongside by using the Circle Merging Quadtree.

The implementation of the CMQ extension for circle packing and the conduction of the user survey was done in the course of a co-supervised Bachelor's thesis [Win18].

Chapter 7 presents a system for the Visualization, Transformation and Analysis (Vat) of spatio-temporal data. This geographical visual analytics system uses the Circle Merging Quadtree as the main method for visualizing point data and, thus, creates a context for the use of this algorithm. The chapter gives an overview of

Vat and details on the exploratory usage by using its interactive user interface Wave. Furthermore, it presents two user studies that show the validity of the approach.

This chapter contains parts of [Bei+17b], [Bei+17a] and [Bei+17c].

Chapter 8 concludes the thesis and presents ideas for future work.

2

The Problem of Visual Point Clustering

In the following, we define the problem setting of *visual point clustering*. With regard to the field of cartography, the problem setting relates to map generalization. Slocum et al. [Slo+09] define map generalization as reducing the amount of information on a map because of scale, mapping purposes, audience or technical constraints. We focus on scale and technical constraints because we aim to provide a scalable method for displaying large point data sets on maps of a certain size. These point data sets reflect discrete phenomena, e.g. observation records of species or points of interests in cities.

Slocum et al. furthermore discuss several visualization variables. For discrete phenomena, they prefer dot and circle maps over choropleth or isopleth maps (cf. Chapter 3). The two latter types of maps show relative aggregates of portions of the map or group parts of the map that share the same values. They claim that this is favorable for displaying continuous phenomena. Furthermore, they see advantages of *proportional symbol maps* in displaying information about raw totals, e.g. the number of animal occurrences in a location. These symbols can be circles (as in our case) but also rectangles or pictographs, e.g. for visualizing the consumption of coffee as coffee mugs of different sizes. However, we do not consider pictographs in our problem setting, but concentrate only on circles. This is in accordance with Forrest et al. [FC85] who found that users are much faster in identifying abstract shapes like circles than iconic shapes. As an example, the prominent mapping service Google Maps[1] encloses icons in circular shapes.

[1] maps.google.com

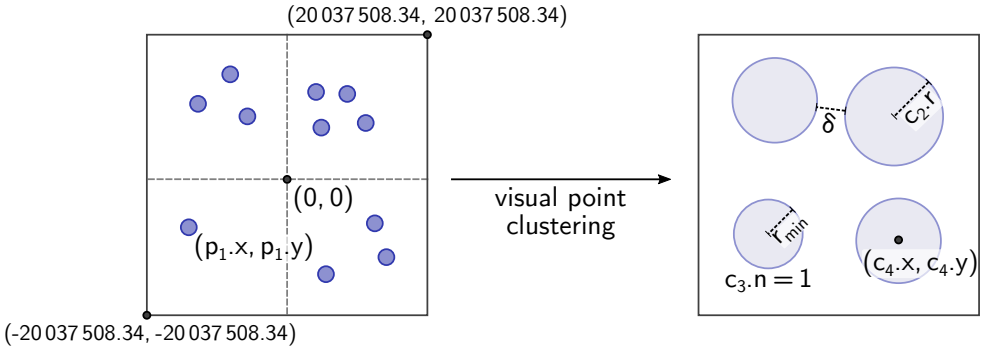


Figure 2.1.: This figure illustrates the definition of a map, points and circles as well as the notion of a minimum radius and inter-circle distance.

2.1. Problem Definition

We consider a two-dimensional coordinate system (cf. Chapter 3) specified by its lower left and upper right coordinate (e.g. Web Mercator with lower left corner $ll = (-20\,037\,508.34, -20\,037\,508.34)$ and upper right corner which equals to $-ll$). Moreover, there is a potentially very large set of points $P = \{p_i = (x_i, y_i)\}$ with $i = 1, \ldots, n$ given, where x_i corresponds to the x-coordinate and y_i to the y-coordinate of a point p_i. Every point represents a discrete object of the real world, e.g. an animal observation or the location of a hotel. Our goal is to compute m clusters and a corresponding set of augmented circles $C = \{c_1, \ldots, c_m\}$ where c_i refers to the i-th cluster. Moreover, we consider $m \ll n$ since we aim for a large reduction of visible information on the map.

A circle $c = (x, y, r, n)$ consists of the coordinates $c.x$ and $c.y$ of its center, its radius $c.r$ and the number of points $c.n$ of its associated cluster. Note that within this thesis, we define a circle as a tuple or vector, but we will access the fields of the tuple by using the corresponding attributes. The circles C should exhibit the following four properties with respect to the input set of points P:

i) The number of circles is unknown in advance, but will typically be much smaller than the number of points.

ii) Each circle represents a cluster, which is a partition of points from the data set.

iii) The circles do not overlap each other.

iv) The circles summarize the underlying point data set in terms of locality, distribution and cardinality. The area of a circle reflects the number of associated points, i.e., a larger circle corresponds to more points than a smaller circle.

Figure 2.1 shows an illustration of the definition of a map, points and circles, coordinates and attributes. The left side shows raw points on a map using the Web Mercator coordinate system. The right side illustrates the output of the aggregation, which is a set of circles. We use the term *visual point clustering* to refer to the problem, as it is a special type of clustering with the specific constraints mentioned above. With regard to cartography, visual point clustering outputs a proportional circle map.

Our goal is to use these circles for visualization on a screen of a fixed width and height given in pixels. Without loss of generality, we focus only on quadratic maps. Of course, circles have to be perceivable and individually distinguishable by users. Therefore, we define two parameters r_{min} and δ as illustrated on the right hand side of Figure 2.1. The parameter r_{min} is the smallest radius we use for the visualization of a cluster with one point. The parameter δ denotes the minimum inter-circle distance. These values should follow the characteristics of the application and the user's screen, e.g. setting r_{min} relative to the font size and δ to 1 px. In the scope of this thesis, we consider these parameters as given. These parameters correspond to research from Jänicke et al. [Jän+12], who also introduced the concept of a minimum radius and distances between circles (they use the parameter ϵ as in density-based clustering, cf. Section 3.4). Also general geovisualization research, e.g. from Slocum et al. [Slo+09], emphasizes the importance of visual variables that include size and spacing.

2.2. Zoom Levels

Map applications are typically dynamic. Users are interested in a certain fraction of the coordinate system (e.g. the bounds of Europe in the Web Mercator coordinate system) at a certain zoom level. We consider zooming into the map as visualizing the same extent of the coordinate system on a larger screen. In our case, we use the common definition of a map with a width and height of $256 \cdot 2^z$ px for zoom levels $z \in \mathbb{N}_0$. This corresponds to the zoom level definition in common map applications like Google, Bing and Open Street Maps [Pic17] as well as comparable research approaches [Jän+12]. Figure 2.2 illustrates the effect of this definition on the size of the map. Note that the sizes are schematic and not pixel accurate. For

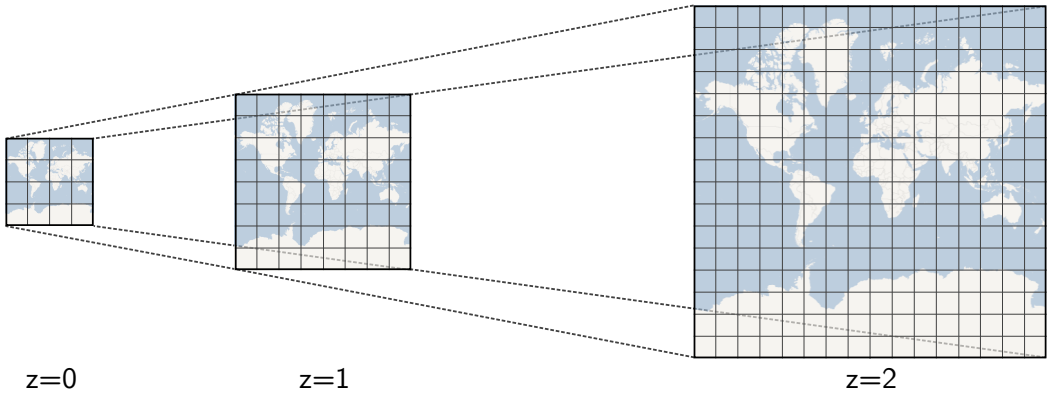


Figure 2.2.: This figure illustrates the effect of different zoom levels.

illustration, we show 4 instead of 256 pixels for $z = 0$. Aside from that, one can see the exponential growth of the map.

Our primary task is to generate a visual point clustering for a given point data set P and a zoom level z. This fits best to an exploratory scenario and ad-hoc map generation. Additionally, we consider generating multiple zoom levels $\{0, \ldots, z_{max}\}$ at once in a setting where we provide a fixed set of pre-computed zoom levels to a user. A reasonable value for the maximum zoom level is, for instance, $z_{max} = 4$. This already results in a map with a resolution larger than Ultra HD.

In real-world scenarios, zooming always leads to restricting the map extent. This would make larger values for z_{max} reasonable. However, for simplicity we only address the case of visualizing the whole world or coordinate reference system, respectively. This is sufficient in order to investigate the effects of different map sizes. We will use many different data sets in our experiments (cf. Chapter 5) to examine different distributional patterns without having the necessity of considering spatial subsets explicitly.

Note that we use the notion of pixels to define the map and the circle sizes. In general, a device screen displays a set of pixels in a specific region, which is referred to as PPI (pixels per inch). Without loss of generality, we consider our pixels to adhere to the web specification[2] of 96 PPI. Thus, for screens that have a very high resolution, it is possible to adapt the minimum circle radius r_{min} as well as the inter-circle distance δ by increasing them accordingly.

[2] www.w3.org/TR/css-values-4/ (accessed December 2, 2018)

2.3. Miscellaneous Point and Circle Attributes

The previous definition of point data only includes the spatial attributes that are relevant for the visual point clustering problem. However, applications often come with miscellaneous attributes about the points. These can be categories as well as numerical or textual descriptions, e.g. a hotel location with the name of the hotel and the number of bedrooms.

In particular in Chapter 6, we will deal with additional attributes that extend the previously mentioned list of (spatial) attributes of points and circles. We refer to these values by their attribute name in the same manner as, for instance, the number of associated points $c.n$ of a circle c.

3

Fundamentals and Related Work

This chapter presents fundamentals and related work of this thesis. It starts with a definition of geographic information systems in Section 3.1. Here, we describe connected fields of research and give a brief introduction to coordinate reference systems for maps. Then, Section 3.2 presents the concept of visual analytics. In particular, we discuss connected fields of research as well as details of spatial information visualization. Section 3.3 contains information of relevant spatial data structures and methods. These are utilized throughout this thesis and serve as a reference. Furthermore, we present related work of aggregating and thinning point data. In Section 3.4, we detail on generic clustering methodologies. We present representative algorithms of several categories that are in particular used in Chapter 4. Finally, Section 3.5 presents generic methods to assess the quality of data clusterings.

3.1. GIS

Geographical Information Systems (GIS) are special kinds of information systems that visualize, transform, analyze and manage spatial data [SW04; HdB09]. A list of points of interest, roads, country borders, satellite images and models of buildings are examples of this data. They are all spatially located on a map and often have additional attributes such as time.

3.1.1. Data Types, Components and Operations

For data management it is necessary to access data from various sources and to cope with different data formats. The main differences constitute in vector and raster data. Vector data are composed of points, lines and polygons. A common data standard is the Simple Feature Access Model [Ope10a]. In this model, points

are two- or three-dimensional coordinates, lines are lists of coordinates and polygons are specified by inner and outer rings (connected lines) that allow arbitrary shapes and holes. Furthermore, there are collections, i.e., sets of points, lines or polygons. Spatial information (e.g. country borders as a collection of polygons) in combination with additional attributes (e.g. the number of citizens) form a feature. In contrast, raster data (also called raster images) represent a discretization of the map space by a fixed grid (cf. Section 3.3.1) of (mostly equal-sized and quadratic) cells. For instance, every cell displays a $100\,m \times 100\,m$ region of the world by a single value, e.g. the temperature of a climate model. Since many relevant data sets on a global scale are very large, e.g. hundreds of gigabytes for a single raster, it is common to store these grids in multiple resolutions that are called pyramids or overviews, for faster access. These formats are stored with various data types. Examples for points are CSV files and the JSON extension GeoJSON [Ope10b]. Raster data is stored, for instance, in GeoTIFF [RR97] or in NetCDF [RD90].

For data access there are standardized interfaces described by the Open Geospatial Consortium (OGC) like the Web Feature Service [Ope10b], Web Map Service [Ope06] and the Map Coverage Service [Ope10b]. The Web Feature Service (WFS) allows accessing vector data and provides data set selection and filtering mechanisms. The Web Map Service (WMS) is used to query raster data from web servers in order to retrieve images that can be displayed directly to users. The Web Coverage Service (WCS) provides means for retrieving geographical data in a processable format, e.g. rasters in GeoTIFF.

For the physical management of spatial data, GIS employ specialized data structures for two- or three-dimensional data. Representatives are grids, quadtrees and octrees [Alu04] as well as R-trees [Bec+90] (cf. Section 3.3 for an overview of spatial data structures). These structures facilitate an efficient data access, e.g. for point and region queries.

GIS also offer functionality for data transformation. Examples are transformations of data formats and data types like the transformation of vector data to raster data and vice versa. A common approach of rasterizing a set of points and polygons [Pin88; HdB09] is coloring the grid cells of a raster that either contain the points or intersect with the polygon. Furthermore, lines and polygons can be transformed to a raster via heatmaps [WF09] and kernel density maps [LH11]. On the other hand, vectorization [Zha+16b; Men03] can be used to transform a raster image into a vector representation, e.g. extracting polygonal regions from a satellite image. Special cases are choropleth maps [Slo+09] in which contiguous polygonal surfaces present regions of aggregated values that are intercomparable, e.g. the number of citizens in countries. Isopleth maps [Slo+09] present regions that share common

values, e.g. contour lines of elevation levels in mountain areas. All spatial data additionally complies with a coordinate reference system, which we will discuss in more detail in Section 3.1.2. Here, GIS also offer functionality for the transformation of data from one coordinate reference system to another. This is in particular necessary for a joint analysis of multiple data sets.

Among the most important goals of GIS is the analysis of spatial data. First of all, this includes the provision of basic operations like filtering data, calculating distances and computing the convex hull of a set of points. Additionally, GIS employ techniques from data mining to find and extract relevant patterns in the data [SW04]. This includes finding correlations, e.g. the influence of weather on the number of visits of points of interest. For this, it is often necessary to combine data sets. Spatial joins [PD96; Bae+07] allow the combination of multiple vector or raster data sets, or a combination of both. This is also used for calculating *map overlays* [SW04], e.g. clipping environmental raster data with polygonal country borders. Aggregations like clustering (cf. Section 3.4) are also employed by GIS. They generate overviews of data or provide additional insights into humanly incomprehensible amounts of data.

Visualization of geographic data is also a key component of GIS. In particular, this reflects in automatic map generalization [SW04]. GIS are responsible for layouting spatial information on maps in a way that users receive a high information value. For raster and vector data, GIS provide colorizations, e.g. with respect to certain class attributes or continuous numeric variables. In addition, GIS provide different symbols for displaying, for instance, point data. Despite the common circular representations, it is generally possible to use individual images that allow distinguishing different types or classes. Different symbols for points of interests on a city map, e.g. house pictograms for hotels and crosses for churches, are a ubiquitous example. The visualization form is also correlated to map legends, which explain the properties of symbols and colors on the map. Furthermore, if there is too much raw data for visualizing it on a map, methods like aggregation (cf. Section 3.3.3) or thinning (cf. Section 3.3.6) are employed.

In addition to the spatial nature of data, the combination with temporal information is often important for geo analysis as well [SW04; KK94]. GIS then answer questions, for example, about the temporal change of certain environmental phenomena. The miscellaneous data represents another dimension of the data. In addition to the map view, also plots like line and scatter plots over time spans are within the toolbox of GIS [ASS02].

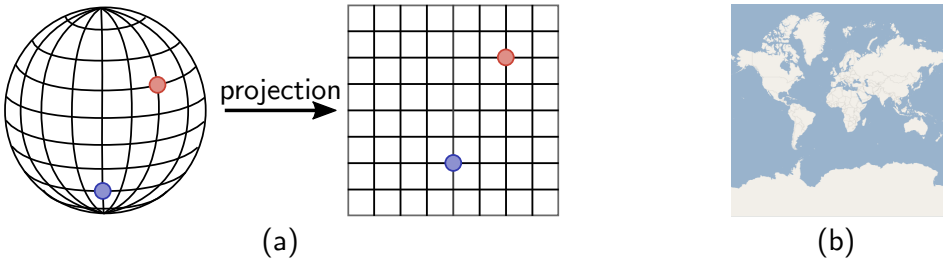


Figure 3.1.: This figure shows the idea of a map projection (a) and a world map in Web Mercator projection (b).

3.1.2. Coordinate Reference Systems

Coordinates in coordinate reference systems (CRS) allow the specification of the position of spatial objects. A special kind of CRS are georeferenced coordinate systems. They have a connection to the earth's surface. Since the earth is neither flat nor completely round, different ellipsoid definitions (also called spheroid or spatial datum) approximate the earth's globe. The most prominent CRS is the World Geodetic System (WGS) [SM98] that was revised in 1984 and is therefore called WGS 84. Its coordinates refer to the horizontal and vertical angles around the center of gravity of the ellipsoid. The coordinate $(0, 0)$ is located west to Africa.

In order to display coordinates that are located on the ellipsoid of a CRS on a flat surface, e.g. a computer screen, it is necessary to project the coordinates onto the two-dimensional plane (cf. Figure 3.1 (a)). Map projections are mathematically defined functions that reduce the dimensionality of the coordinate. This reduction unfortunately always causes an error, which is characterized by a loss of information due to distortion. However, it is possible for a projection to be either area preserving or angle preserving, which is also called conformal [Mal13]. In the former case, all relative areas of spatial objects are maintained. The latter case retains all local angles. This means that a straight line stays a straight line and a 90° curve still remains a 90° curve. Hence, for different scenarios it is useful to choose a map projection that induces the smallest error. Web Mercator [Bat+14] (cf. Figure 3.1 (b)) is a prominent projection that is used in many map services. It is almost angle preserving since it induces a small error, but it is very simple and efficient to compute. A disadvantage of this projection is that land masses in the north and south appear much larger than land masses in the middle of the map, e.g. Antarctica appears larger than Africa, although it is smaller.

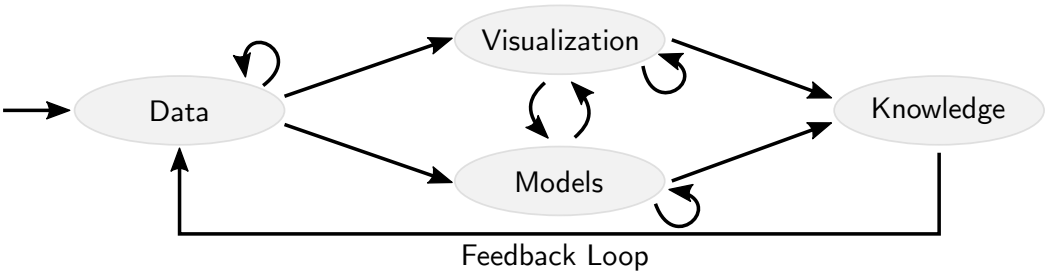


Figure 3.2.: This figure shows the visual data exploration cycle as presented by Keim et al. [Kei+08b].

3.2. Visual Analytics

Visual analytics is a scientific discipline that is concerned with analytical reasoning and the provision of visual interfaces for its support [CT05]. The visual interfaces help to improve the understanding of the incorporated data [CMS99]. Keim et al. [Kei+08b] describe the division of labor in an analytical task between humans and machines. The idea is to use combined strengths in order to compensate for the weaknesses on each side. For instance, humans are extraordinarily good at recognizing shapes and visual relationships, but in comparison, they are rather slow in calculating statistics for millions of values. Keim et al. furthermore describe a visual data exploration cycle (cf. Figure 3.2) in which data leads to interconnected visualizations and models, and finally leads to knowledge. This knowledge can also be transferred back to the selection of data sets for further investigations [Che+09].

Visual analytics is often associated with data-driven research. In this methodology, the exploratory analysis of data leads to insights without having a concrete hypothesis in mind at the beginning [Kei+08a]. Additionally, it supports the rapid testing of hypotheses and the retrieval of immediate feedback from visual analytics systems. Tools of different scientific fields help to investigate data. This reduces the effort for humans to search for data and to recognize patterns in data. Furthermore, these systems facilitate finding relationships between data objects and abstract the handling of data.

3.2.1. Fields of Research

As previously mentioned, visual analytics constitutes an intersection of different research fields [Kei+08b]. One major field is the scientific visualization of data in

combination with human recognition. Human Computer Interaction (HCI) provides guidelines for designing intuitive user interfaces and for employing working interaction styles that improve the connection between the visual analytics system and the human user. In particular for spatial data (cf. Section 3.2.2), the visual aspect plays a key role [AAG03]. But also scientific visualizations of data from other domains are necessary to facilitate the understanding of users. Furthermore, the quality of the visualization and the acceptability of the presented results need to be considered.

Data management is another important research area in the visual analytics context [Kei+08b]. One of the goals is to be able to analyze large amounts of data. For this task fast data access times, scalability and latency are properties of utmost importance. With respect to latency, Lui and Heer [LH14] investigated its impact on exploratory visual analysis. Systems that are able to provide results (refresh times) in about $100\,ms$ are considered to offer a *continuous perception* to the user. This agrees with findings of Heer and Shneiderman [HS12] that prefer computations at "rates resonant with the pace of human thought". Findings show that a low latency in user interactions leads to better analytical performance and encourages insight generation [LH14]. However, there are operations that are more or less sensitive to delays in responses. For instance, users expect an immediate feedback of brushing and linking operations [CMS99]. These operations benefit from efficient algorithms and systems that utilize aggressive caching. On the other hand, zooming is less sensitive, and thus users are more tolerant to small delays. Chen et al. [Che+09] emphasize that the visualization of response time indicators is also part of information visualization. In general, additional delays of $500\,ms$ have a negative impact on the user's performance by using a system [LH14]. Users find a system unusable if delays are too long and there is no feedback. Furthermore, slower systems dampen the user's motivation of doing their tasks. This effect is not deliberately recognized by users, but influences their behavior measurably.

Data mining is the third field of research that has an impact on visual analytics [Kei+08b]. It provides tools for browsing data and for investigating interesting subsets. Means for finding correlations facilitate the formation and verification of hypotheses. Cluster analysis (cf. Section 3.4) is a crucial analytical tool that on the one hand facilitates structuring data and on the other hand is applicable for visualization. For the latter it can be used as preprocessing to reduce the amount of data a human user has to inspect. Generally, data abstraction is important for visual analytics. Chen et al. [Che+09] emphasize the importance of finding an optimal abstraction level.

The general approach to visual analytics is described by the famous information

seeking mantra from Shneiderman [Shn96]. It states that it is important to first provide an overview to the user, then to present options like zooming and filtering, and finally to provide further details on demand. Keim et al. [Kei+08a] extended this mantra by suggesting doing an analysis of the data first. They propose to show important subsets or findings subsequently and to provide means for further analyses, zooms into the data and applications of filters. Finally, this methodology also requires the provision of more details of the results on demand. This extension takes into account that big data requires more sophisticated analytical methods in order to provide overviews to the users. Shrinivasan et al. [SvW08] define the sense-making loop as a visualization of the visual analytics process. A user often has no clear goal at the beginning of a data analysis [LH14]. Moreover, this process is often open-ended since one finding may lead to additional analytical tasks and results.

3.2.2. Spatial Visualization

With respect to the topic of this thesis, we provide some more details regarding information visualization and in particular spatial visualization. Card et al. [CMS99] claim that the purpose of visualization is to provide insight and not just to create pictures. This underlines the importance of this topic. Ackoff [Ack89] presents a distinction between data, information and knowledge. He defines that data itself is just a collection of raw symbols and that information provides answers to questions of *who*, *what*, *where* and *when*. In addition, knowledge provides answers to *how* questions. A good visualization facilitates at least the provision of information to the user. In general, big data leads to the information overload problem [Kei+08b]. To solve this problem, it is of utmost importance to aggregate data and present it in an appropriate way.

In geography, data is often investigated with regard to topographical or topological aspects. Topography describes the investigation of the land surface and its features and relates more to raster data (cf. Section 3.1). In geography, topology refers to the spatial relationships between different objects such as points and polygons. Statistical indicators like plots are miscellaneous methods to investigate spatial data.

Research in the field of spatial visualization has developed several guidelines that should be followed. Munzner [Mun14] questions the unjustified use of three-dimensional visualization and insists on using two-dimensional visualizations whenever useful. Furthermore, she emphasizes that occlusion hides information and that data should be presented in a way that reduces cognitive load for users. For instance, side-by-side views like maps and accompanying plots are in favor over

animations. MacEachren [Mac04] emphasizes that the number of colors of symbols on maps should be limited in order to preserve distinguishability. His findings show that 10 different colors allow 98 % color discrimination while the rate drops to 72 % for 17 colors. Furthermore, he points out that simple abstract shapes like circles are more easily recognized than complex shapes or icons. Slocum et al. [Slo+09] present findings that nearby circles on maps are considered to belong to a group and that similar sizes do not interfere with this impression. Kraak and Ormeling [KO13] also point out that points on a map should not always be visualized as simple dots, but rather as circles with varying sizes if reasonable.

Ellis and Dix [ED07] propose a set of eight criteria concerning clutter removal in information visualization. These criteria are also applicable to spatial visualization. A good visualization *avoids overlap* such that users can identify patterns and understand data better because clutter leads to information loss. In addition, they require that users *can discriminate points*, which means introducing a noticeable boundary between map elements. Furthermore, the criteria emphasize that it is important for data to *keep spatial information* and that data *can be localized*. Both aspects are important for geographical data where users need to recognize locations, e.g. countries or mountain areas. A visualization that *is adjustable* allows users to change aspects of the displayed results. Another key aspect is the requirement that the visualization *is scalable* such that it can handle large data sets. The penultimate criterion demands that a visualization technique *can show point attributes* because they consider miscellaneous attributes to be important as well. The last criterion is the requirement that users *can see overlap density*, so that a loss of information can be reduced when overlaps occur.

3.3. Spatial Data Structures and Methods

This section discusses spatial data structures as well as methods with the focus on two-dimensional point data in alphabetical order. It starts with the grid data structure in Section 3.3.1 for basic indexing and querying of spatial data. Then, it presents histograms in Section 3.3.2 as a tool to assess the distribution of data. Afterwards, it presents point aggregation techniques in Section 3.3.3. Subsequently, it presents quadtrees in Section 3.3.4 as a more sophisticated spatial indexing method. In Section 3.3.5 we present space-filling curves as a method to linearize two-dimensional indices. Penultimately, this section discusses the concept of thinning a set of points for visualization in Section 3.3.6. Finally, the section presents Voronoi diagrams and Delaunay triangulations in Section 3.3.7.

Figure 3.3.: This figure shows a fixed grid data structure (a) and a rectangular query to the grid (b).

3.3.1. Grid

A basic sequential (list) storage of data is easy to implement, but has disadvantages regarding the efficiency of spatial queries, e.g. rectangular queries. This requires to scan the entire list for each query. This can be improved by partitioning the data into a limited number of subsets. Hence, for a query, only certain subsets need to be examined. A *fixed grid* [Sam04] stores the data in equally sized cells. If the data domain is known in advance, it is possible to subdivide the space accordingly, e.g. for two dimensions in the spatial processing context. The common shape of such cells is a square, but it is also possible to store, for instance, rectangles, triangles or hexagons [Sco15]. Every cell refers to a list that stores the data that is contained in the cell. Figure 3.3 (a) shows an exemplary grid with 64 cells that stores spatial point data. The method works best if the data is uniformly distributed. A drawback of grids is their sparsity in case of non-uniform data. For example, if data points fall only into the border regions of the grid, it is still necessary to store the whole grid structure. This can be improved by storing only the non-empty cells in a hash map by using the two-dimensional index of the cell as input to the hash function. Furthermore, there can be large differences in the amount of data stored in single grid cells, e.g. for dense hot spots.

Since the grid cells are of equal size, the lookup of the corresponding grid cell can be efficiently done in constant time [Sam04]. Hence, for insertions and deletions the runtime only depends on the underlying data structure, e.g. a linked list. For rectangular range queries (cf. Figure 3.3 (b)) it is necessary to compute the cell indices of the rectangle vertices and traverse all cells in between. For radius queries, it is necessary to compute the bounding square and use the previous method. Then all cells are first checked with a square-circle intersection, and later all data items within the qualifying cells are checked for containment. Kanth and Singh [KS99] showed a lower bound for two-dimensional range queries of $O(\sqrt{n}+r)$ for r elements in the result set.

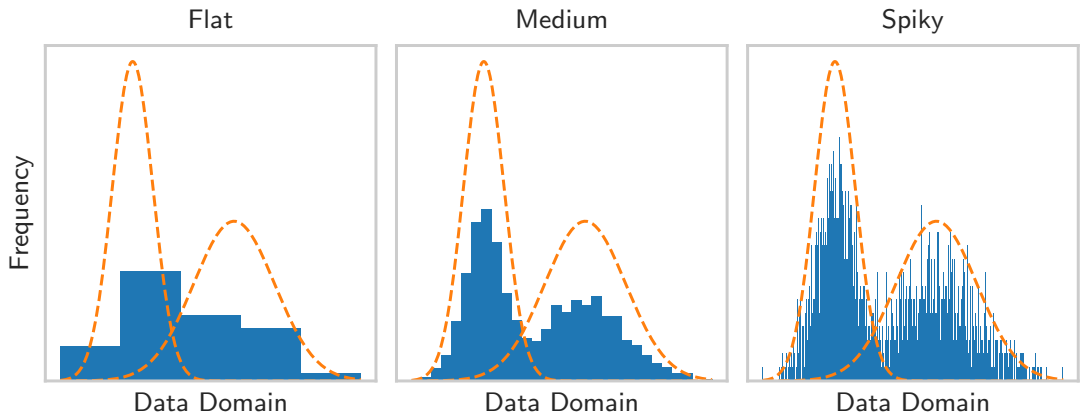


Figure 3.4.: This figure shows three histograms with differing numbers of buckets for multimodal normal distributed data.

Grids can be used for data aggregation as well. The STING method [WYM97] uses grids of multiple resolutions and stores data statistics in its cells rather than the data itself. Since computing the grid index requires constant runtime, it is possible to compute the grid in $O(n)$ time for n data points and a single resolution. This method presents a connection to two-dimensional histograms (cf. Section 3.3.2), which basically store count information for each cell.

3.3.2. Histogram

Histograms [Sam04] represent a method that allows for an overview of the distribution of a data set. The idea is that the data follows a latent probability distribution and that the method tries to estimate the probability density function of the underlying continuous random variable. The goal is to assess whether the distribution is, for instance, uniform, symmetric or skewed. Furthermore, it is possible to assess uni- or multimodality as well as other characteristics.

The method subdivides the data domain into parts that are called bins or buckets. Then it counts the number of data items that fall into the buckets. The height of the histogram bucket, which is represented as a bar diagram, is then proportional to the data frequency. The common variant of a histogram has buckets of uniform width (*equiwidth*), but there are other variants that have a uniform height (*equiheight*) and modify the bucket width in a way that the bucket area reflects the frequency.

The correct choice of the number of histogram buckets is crucial for facilitating a meaningful interpretation [Sco15]. For instance, too few buckets show a flattened out structure of the distribution (the extreme case of one bucket has no expressiveness other than presenting the cardinality of the data set) and too many buckets produce a very spiky representation with lots of empty buckets. Figure 3.4 shows an example of three histograms that visualize the same data with differing numbers of buckets. The blue histogram bars present the data frequency and the dotted orange line represents the actual data distribution. Apart from domain knowledge, there exist general rules or formulas to calculate a suitable number of buckets k from the number of data points n. Two examples are the *square root choice* [Mac11] and *Sturge's formula* [Stu26]. The former calculates $k \coloneqq \lceil \sqrt{n} \rceil$ and tries to balance the number of buckets with the number of data items within the buckets. The latter calculates $k \coloneqq \lceil \log_2(n) \rceil + 1$ and generally works well with normally distributed data, but also poorly with non-uniform data in practice.

The height of the histogram is often normalized. In this way, the histogram can represent a discretized version of a probability density function. In database management systems, histograms are good means for data statistics [Sam04]. In particular, they are used for query optimization by estimating the cardinality of ranges and areas in data.

3.3.3. Point Aggregation

Since visualizing point data is a crucial operation for spatial data analysis (cf. Section 3.2), there are techniques for providing an aggregated overview of data sets. These techniques present a mixture of data generalization and data mining and are tools that allow users to assess the distribution of large data sets. In the following, we present a number of representative methods. Note that we discuss thinning as a generalization method in Section 3.3.6.

Jänicke et al. [Jän+12; JHS13] developed *GeoTemCo*, which is a spatial data aggregation technique that exploits Delaunay triangulations (cf. Section 3.3.7) [dBer+08] for finding close points or circles that represent aggregates of points. They organize the edges of the triangulation in a priority queue. Their similarity function incorporates the density of the two vertices of the edges by introducing circles with centers at the coordinate of the vertices and radii that are proportional to the number of reflected points. The algorithm then iteratively finds the edge with the highest similarity and reduces its vertices to a single vertex. This vertex then represents all points from the former two vertices of the edge. The algorithm continues until the circles of the edges' vertices do no longer produce overlaps.

This requires updating the triangulation as well as the priority queue for each step. The runtime of the algorithm is $O(n \cdot log(n))$.

Bereuter et al. [BW13] describe a toolbox of different data aggregation techniques in order to reduce the amount of data in mobile scenarios. They use a quadtree (cf. Section 3.3.4) as data structure for storing the points and then apply different algorithms to create aggregations or simplifications. These algorithms range from using the coordinates of the filled quadtree nodes to displacing circles in order to remove overlap. Their centroid method calculates the centroids of the quadtree's nodes with respect to a minimum circle size that determines the maximal node depth of a query. Each centroid reflects one circle in the result set where the radius is determined by the number of points in the associated tree node (and its children). A user-defined factor defines a proportional scaling of these radii.

Grid-based techniques [GB17; LJH13] rely on dividing the space into equally sized partitions and binning the points. These methods cannot locally adapt the spatial extent of the partitions and, hence, their cell sizes do not reflect the density. Furthermore, they use different colorings for the grid cells [LJH13] or use larger cells in which they place symbols of different sizes [GB17].

Zhang et al. display point data as overlapping circles, but map the points onto another spherical surface for comparison. Other methods [Jän+12; Zha+16a] focus on comparative visualization of multiple data sets. Jänicke et al. [Jän+12] use groups of circles (circle packing [Can+07]) to represent groups. Ghanem et al. aggregate multiple data sets at once and display pie charts instead of circles [Gha+14]. These pie charts depict the fractional occurrence of the individual data sets.

3.3.4. Quadtree

The quadtree [Alu04; Sam06] is a spatial index structure for two-dimensional data. It is straightforward to generalize it to higher numbers of dimensions, e.g. the octet tree for three-dimensional data. The idea is to split the data space at each level of the tree into four quadrants, which can be referred to by using the geographical regions northwest, northeast, southwest and southeast.

The *point quadtree* [Alu04] uses data elements to define the quadrants (data partitions) at each tree level. Each quadrant stores exactly one item. This procedure is disadvantageous for external storage because of the large number of pointers that lead to random reads. Figure 3.5 (a) illustrates a point quadtree for a set of points. Although the height h of the tree is $O(n)$, it is possible to sort the data and use the

Figure 3.5.: This figure illustrates the quadtree data structure. The left side (a) shows a point quadtree and the right side (b) shows a region quadtree with a node size of one point. Both quadtrees store the same set of points.

medians recursively to receive good split points. This leads to a logarithmic height of the tree. Insertions and point searches are of $O(h)$ time complexity. Deletions are complicated to handle since the data points themselves define the quadrants and reorganizations are necessary [Alu04].

The *region quadtree* (also known as *point region quadtree*) [Alu04] divides the data domain in four equally sized quadrants at each tree level. This means that the split points are data independent and the quadrants can be referenced by two bits per level (cf. tries [Sam04]). For region quadtrees it is possible to store a fixed number of items in a tree node (block of data). Splits are performed whenever the data blocks overflow. Since overflows can also occur after splits when all items fall again into the same region, the degenerated tree does not have a height bound (and thus no space limit) that only depends on the number of points n. However, region quadtrees have a height bound of $O(n \cdot log(\frac{D}{s}))$, where D represents the length of the root cell and s indicates the smallest distance between two points in the quadtree [Sam84]. As for the point quadtree, insertions have $O(h)$ time complexity. Deletions are straightforward to implement for region quadtrees and the same runtime applies here. Underfull regions can simply be discarded.

Point searches examine one quadrant per tree level via a point in rectangle test and, hence, have a runtime of $O(h)$. Note that it is necessary to have a convention for determining to which quadrant a point belongs that is exactly on the border. Rectangular range queries have an expected runtime of $O(log(n)+|C|)$ for n points and $|C|$ result cells if the query region is much smaller than the total data domain. In this case, since the size of the incorporated tree quadrants (tested via rectangle-rectangle overlaps) decreases exponentially, the query can efficiently find the cell that covers the result in logarithmic time. Then it requires $O(|C|)$ time for accessing all cells C that contain the query result. Spherical queries can be handled sim-

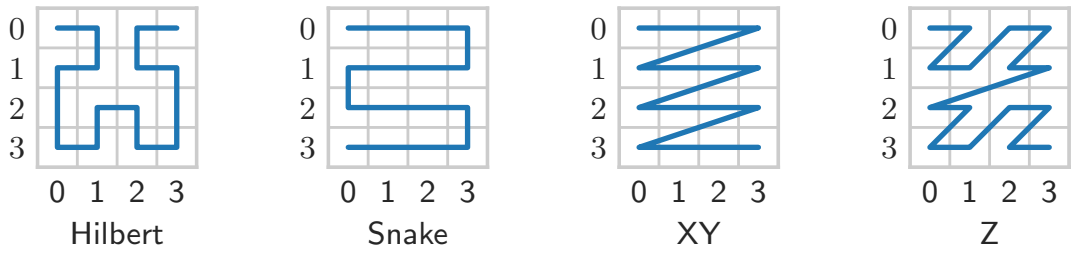


Figure 3.6.: This figure shows four space-filling curves for linearizing the indices of a two-dimensional grid.

ilarly like rectangular range queries (cf. Section 3.3.1).

Quadtrees can as well be used for storing more complex structures like lines or polygons [Sam04]. In this case, the vertices of the structure are either stored in different quadtree nodes (*edge quadtree*) or each node that intersects with the structure stores a pointer to the structure (*MX quadtree*). For example for lines, the intersection of quadrant (rectangle) and line determines if a pointer is stored in the quadrant. In case of splits, the check has to be recomputed for the new, smaller quadrants.

3.3.5. Space-Filling Curves

Index structures often discretize multi-dimensional data domains in order to access certain regions efficiently. In a grid, we refer to these regions as cells of equal size. The traversal or storage layout of such cells is not unambiguous. We limit this discussion to the two-dimensional domain since it is easier to understand and more relevant for this thesis. Here, in each step, it is possible to choose among all eight adjacent cells or to jump to an arbitrary, previously unvisited cell. Space-filling curves [Bad13; SW04] provide a linear ordering of cells.

Figure 3.6 depicts four variants of space-filling curves for a discretization of the data domain into 4×4 cells. Each cell has an index on the x and y axis, e.g. $(0, 0)$ is the starting point on the upper left cell for each curve. Note that we flipped x and y such that the depictions of the curves better represent their characteristics. The *Z curve*, also referred to as Morton ordering, is the most prominent representative. It uses the bit representation of the x and y values and calculates a new value by using bit interleaving. For instance, the two 4-bit representations of $(2_{10}, 1_{10}) = (0010_2, 0001_2)$ is $9_{10} = 00001001_2$. The *XY curve* corresponds to a default data layout for storing an array of arrays contiguously on disk. Here, we fix the index

of one dimension and iterate over all indices of the other dimension. Then, the index of the first dimension is increased until all positions are visited. Since the XY order makes large jumps after traversing one dimension, the *snake curve* only iterates over adjacent cells. In contrast to the XY curve, it iterates in a forward fashion over the second domain if the index of the first domain is even. In the other case, it iterates backwards. By iterating this way, the distance to the next cell is always minimal. The *Hilbert curve* combines the minimal cell distances of the snake curve with lower index differences of adjacent cells. For the snake case, for instance, the change from $(0, 0)$ to $(1, 0)$ results in a large gap between the indices. The Hilbert curve provides more locality in this sense. It is calculated by recursively applying the basic form of the structure to the positions in the grid and additionally uses rotations.

All the provided space-filling curves are self-similar [Bad13]. This means that the small versions in Figure 3.6 represent puzzle pieces for larger grids such that they provide the same structure on a larger scale. Hence, a grid of size $2^k \times 2^k$ consists of four $2^{k-1} \times 2^{k-1}$ grid curves [SW04]. This means that they keep their properties regardless of the number of discretized steps in the data domain.

3.3.6. Thinning

A common strategy for visualizing large amounts of point data on a screen is thinning by sampling the data [PCM16; Wan+15; Wan+12; Arb93]. This is in contrast to aggregating data points (cf. Section 3.3.3) and displaying them as circles of different sizes. Thinning is a technique that yields a data reduction and can reduce occlusion.

Simple random sampling [Arb93] uses basic sampling strategies like selecting every item with the probability $\frac{m}{n}$ for retrieving approximately m items from n items in total. If it is required to select exactly m items, it is possible to employ reservoir sampling [Vit85]. Here, we select the first m items into an intermediate list and then replace items of this list for all subsequent items with a probability of $\frac{m}{i}$. The value i represents the index of the i-th item in the data. The replacement among the m items in the intermediate list is done uniformly at random. As a result, each item has the same chance of being placed in the list. The advantage is that the number of items does not need to be known in advance and the procedure only requires a single pass over the data. A practical speed improvement can be achieved by directly calculating the next element that is swapped and then jumping directly to this element or skipping all elements in between, respectively. Since this method only ensures randomness in data selection, it is likely that there

are many nearby elements for dense areas in the data. Thus, this does not prevent elements from overlapping in dense regions.

Systematic sampling [Arb93] can be applied when the data is ordered uniformly at random. Then, instead of choosing each item with a certain probability, it is possible to calculate skips of size $\frac{n}{m}$. This is much faster in practice. If the data is not randomly ordered, but has inherent patterns, this method provides no randomness at all. If the data has certain subgroups and the sampling should consider each of these groups, *stratified sampling* [Wan+12] can be applied. Here, the previously mentioned techniques can be applied to each stratum (subgroup).

A recent work in this field is the thinning approach of Sarma et al. [Sar+12]. Their method heavily depends on a hierarchy or an importance function, respectively, to decide which data to keep and which to leave out. It is suitable, for instance, for displaying cities on a map where we expect to see capitals first, then big cities and finally smaller towns in higher zoom levels. They use a spatial tree index structure (e.g. a MX quadtree, cf. Section 3.3.4) and store the data in predefined cells (dependent on the zoom levels). The algorithm assigns each data item an ordering index, based on its importance in combination with a random variable, and stores the data elements on tree levels based on their ordering. The root node thus contains the most important objects and the leaf nodes the least important ones. During a tree traversal, the algorithm first considers earlier seen elements up to the sample size. In addition, their sampling complies with the quality criteria visibility, zoom consistency and adjacency. Visibility constrains the maximum number of visible items per region. They additionally use the term maximality, i.e., returning as many elements as possible up to the visibility limit. Zoom consistency ensures that items at zoom level z are also visible at all zoom levels $z' > z$. The adjacency is important for data other than points, e.g. polygons. Here, if a polygon is visible in a cell, but spans across other cells as well, it has to be also visible there. For data other than points, they prove that the problem is NP hard and provide an integer program as well as a greedy algorithm that solves the problem more efficiently.

Guo et al. [Guo+18] present a similar sampling technique that also complies with a zooming and panning constraint which defines that visible objects should remain visible after zooming in or moving the map. Furthermore, they introduce a different concept of a visibility constraint. Here, sampled objects should have at least a minimal distance to each other in order to remove occlusion. Furthermore, they introduce a representative constraint. It ensures that all sampled items maximize the similarity to the other values of the region they represent. They prove that this problem is NP hard and provide a more efficient greedy algorithm with

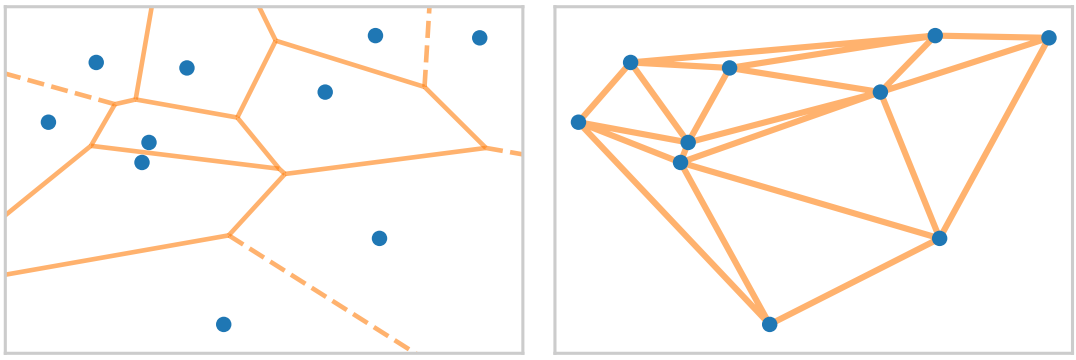


Figure 3.7.: This figure shows a Voronoi diagram of a set of points (a) and the corresponding Delaunay triangulation (b).

an approximation ratio of $\frac{1}{8}$. Although this approach reduces occlusion, the use of a minimum distance between the sampled elements weakens the indication of whether the regions are dense or rather sparse. Furthermore, it is not trivial to choose the sample size.

3.3.7. Voronoi Diagrams and Delaunay Triangulations

Given a set of points on the two-dimensional plane, a Voronoi diagram [AKL13] divides the plane into cells such that each cell is a convex polygon and in each cell is exactly one point. The line segments of the cells are placed in such a way that the two nearest opposite points (centers) are exactly equidistant to the line. This means that within a Voronoi cell, all potential positions are closer to the cell's center than to any other cell center. Figure 3.7 (a) shows an exemplary Voronoi diagram of a set of points.

The Delaunay triangulation [AKL13] describes the computation of a triangle mesh from a set of two-dimensional points. Figure 3.7 (b) shows the resulting mesh of a Delaunay triangulation of a set of points. Here, all points are connected to each other such that they (1.) form triangles and (2.) that no other point lies in the circumcircle of the triangles (the triangle vertices are on the perimeter of the circumcircle). This produces triangles that do not have very narrow angles. Hence, Delaunay triangulations are beneficial for computations because they are not susceptible to numerical rounding errors.

The Delaunay triangulation is a duality of the Voronoi diagram [AKL13]. The triangulation can be computed by connecting all points that share an orthogonal line segment in a Voronoi diagram (cf. Figure 3.7 where (a) and (b) are computed on the same set of points). In contrast, a Voronoi diagram can be computed from a Delaunay triangulation by introducing orthogonal line segments (half-planes) between all triangle edges. Since the number of triangle edges is linear in the number of points, this computation is possible in linear time [AKL13; dBS04]. Note that the Delaunay triangulation is not limited to two dimensions as there are existing generalizations. However, we restrict this discussion to the two-dimensional case and the bounds are only valid for two dimensions.

It is possible to compute the Delaunay triangle mesh in $O(n \cdot log(n))$ time [dBS04]. The basic edge-flipping algorithm builds first an outer shell (triangle) of the points. Then it iterates over the points and connects them to the triangle vertices such that it creates three new triangles. It then uses the circumcircle for every new triangle to test for every adjacent triangle if the opposing point is in this circle and, thus, violates the angular property (b). If it violates the property, the algorithm flips the edges of the two adjacent triangles and continues testing until all potentially affected triangle pairs are valid. This algorithm has a worst-case time complexity of $O(n^2)$, but an expected runtime of $O(n \cdot log(n))$ if the points are visited in random order. The randomness guarantees an upper bound of necessary edge flips. It furthermore requires using a tree structure of triangles for efficient lookups. In addition, a divide-and-conquer algorithm [CMS98] exists that recursively merges triangulations of subsets of the data. The S-Hull (sweep-hull) [Sin16] algorithm uses both a sweep line and a beach line to calculate the triangulation. Both algorithms have a worst-case runtime of $O(n \cdot log(n))$, but are more difficult to implement and do not allow incremental updates.

3.4. Clustering

Clustering or cluster analysis describes the process of grouping a set of data items into multiple groups [HKP11]. These groups are called clusters and the goal for a clustering algorithm is to find clusters that fulfill certain properties. The most important characteristic is that items within a cluster should be similar. To define similarity, one has to provide a similarity or distance function, respectively. Algorithms try to maximize the *intra cluster similarity*. This means that within a cluster the pairwise similarities of items should be as large as possible. In addition, algorithms aim to minimize the *inter cluster similarity*. This means that the similarities of items that are in separate clusters should be as small as

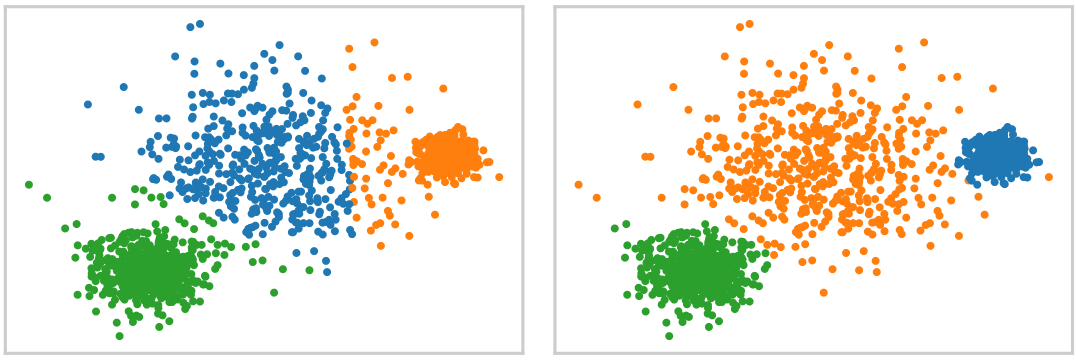


Figure 3.8.: This figure shows two clusterings of a data set generated by the combination of values from three normal distributions with different variances. The left clustering (a) is computed using k-means and the right clustering (b) is computed using EM.

possible. This discrimination leads to a clear separation of clusters from each other.

In the following, we present the four basic kinds of clustering algorithms. We first of all start with *partitional clustering* that tries to divide the data items into a fixed number of clusters. Then we present *distributional clustering* that defines clusters based on probability distributions. Thereafter, we present *hierarchical clustering*, which builds a hierarchical tree of cluster assignments. After this subsection, we insert a subsection that discusses the choice of the number of clusters as user parameter. Finally, we discuss *density-based clustering* that defines the notion of density as similarity measure and does not require a fixed number of clusters to be specified beforehand.

3.4.1. Partitional Clustering

Partitional clustering specifies a class of algorithms that receive the data items and a number of clusters as input, and divides this input into partitions based on a similarity or distance function. The most prominent method in this class is k-means [Llo82]. It basically forms Voronoi-shaped (cf. Section 3.3.7) clusters where the means refer to the centers of the Voronoi cells. Figure 3.8 (a) shows an exemplary k-means clustering of a set of points in the form of a combination of three normal distributions with different variances. The basic algorithm initializes

the means with random values in the data domain. Then, it assigns every data item to its closest mean so that clusters are formed. For each cluster, the algorithm then calculates a new mean by computing the centroid of the cluster items. The algorithm repeats these steps as long as the assignment of items to means (i.e. clusters) changes. This implies that it stops at a local minimum. In practical applications, it turned out that this algorithm converges very fast. In theory, however, researchers proved k-means to be NP hard [MNV12]. They were able to reduce k-means to the 3-SAT problem.

The algorithm k-means++ [AV07] tries to improve the practical runtime of k-means by changing the initialization of the means. Experiments show significant decreases in runtime of several orders of magnitude for real-world data sets. The method iteratively chooses one mean after another from the data items. It starts with one mean by selecting the first data item uniformly at random. The remaining means are chosen at random with a higher probability for data items that are far away from any already chosen mean. After initialization, the algorithm continues with the basic k-means algorithm.

There are also other algorithms like fuzzy C-means [HKP11] that do not enforce a strict partitioning of the data. This helps to overcome the problem that k-means is prone to outliers and items between clusters. Here, an additional weight vector w and the cluster mean vector C are optimized with respect to inter and intra cluster distances. Like k-means, the algorithm stops at a local minimum. The data items are, however, associated to every mean with a degree in $[0, 1]$ so that overlapping and fuzzy cluster borders are the result.

Since the calculation of a mean of data items requires the data items to be vectorial, purely metric data cannot be clustered with k-means. Examples are the comparison of complex items, e.g. polygon similarity or differences in textual documents. Methods that can handle metric data are k-medoids and k-modes [HKP11]. The former one uses the metric distance function and the latter one is a frequency-based method that replaces the mean calculation by a mode calculation. The latter one can thus be used for categorical data. The basic algorithm for k-medoids is Partitioning Around Medoids (PAM). Here, the initial medoids are selected uniformly at random among the data items. Then, the algorithm assigns all items to their closest medoid. Until there is no further improvement, PAM tries to swap the medoids with each of the remaining data items. The improvement is calculated by aggregating over the distances between the data items and their medoid, which optimizes the intra cluster distance. The result is also a local optimum. In practice, k-medoids is slower than k-means. Additionally, it is possible to create a fuzzy assignment for k-medoids similar to fuzzy C-means.

The k-means approach is only able to separate clusters that are linearly separable [DGK04]. Similar to support vector machines [Bis06] one can use the kernel method to transform the data items into a higher dimensional space. This makes the approach ubiquitously applicable.

3.4.2. Distributional Clustering

Distributional clustering follows the approach of using probability density functions (pdfs) instead of representative points or hard cluster assignments. The most prominent representative is Expectation Maximization (EM) [Bis06]. It tries to find the maximum likelihood of a multimodal probability distribution that models the latent data distribution. EM treats the data items as observations and the goal is to be robust against outliers and missing values. For initialization, EM uses random parameters for each of the k pdfs. The algorithm then calculates the probability density for the data items with respect to the k pdfs as a second step (E step). As third step, the algorithm calculates the empirical mean and variance of the data items for each pdf (M step). The probability density is used as weights such that data items with a large density value contribute more to a pdf and vice versa. EM repeats the last two steps until it converges to steady pdf parameters. Figure 3.8 (b) shows an exemplary EM clustering of a set of points (cf. Figure 3.8 (a) to compare the results of EM and k-means).

EM runs much slower than k-means in practice. As an advantage, the result corresponds to a mathematical model of the data distribution. The algorithmic result can represent the data best if the latent distribution fits to the used probability function, e.g. a Gaussian pdf. EM can use a covariance matrix that allows different forms of the probability distribution along the data dimensions. Furthermore, it allows for rotations. Problems can occur if one of the pdfs only contributes a very small density for all data items [Bis06]. Then, as this basically corresponds to an empty assignment, it is hardly possible to calculate the mean and variance parameters. This leads to a non-singular covariance matrix. A solution is then to reinitialize the vanished pdf with random values.

3.4.3. Hierarchical Clustering

Hierarchical clustering decomposes the data items into a hierarchical structure of connections [HKP11]. There are two options to create these hierarchies: top-down or bottom-up. The top-down method is called divisive clustering. It starts with the full set of data items and recursively partitions this set into multiple subclusters

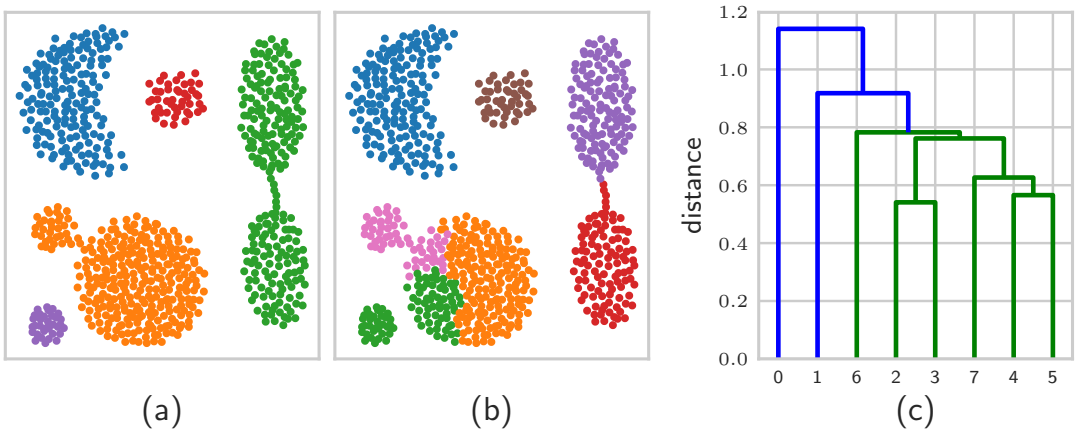


Figure 3.9.: This figure shows two clusterings of the *aggregation* data set [GMT07] that are generated by agglomerative clustering with single linkage (a) and complete linkage (b). Furthermore, the right diagram (c) shows a dendrogram of single linkage clustering.

based on a distance function. Since there are 2^n possibilities to split n data items at each step, the approach is usually very expensive. This leads to algorithms like DIvisive ANAlysis (DIANA) [HKP11], which use heuristics to estimate the split points. The bottom-up method is called agglomerative clustering. AGglomerative NESting (AGNES) [HKP11] is an algorithm that starts by placing every data item into its own cluster. It then iteratively merges the clusters with the smallest distance until all items are in the same cluster. Figure 3.9 (a) and (b) show two exemplary clusterings of agglomerative clustering. Each merge step is stored in a tree data structure that is called dendrogram. Figure 3.9 (c) shows such a dendrogram of an exemplary clustering. A connection (edge) of two clusters in the dendrogram is an indicator of a merge. The vertical length of the edge then indicates the distance between the merged clusters. In order to get a clustering with k clusters or clusters with a certain distance or similarity, it is necessary to virtually cut the tree horizontally to retrieve the clustering. The runtime of agglomerative methods is $O(n^3)$ in general, but there are $O(n^2)$ variants for specific cases (e.g. SLINK [Sib73]).

In the description of the hierarchical methods, we used the notion of a cluster distance instead of an item distance. The way how to calculate this distance is described by the *linkage* method in this context. Note that the linkage still requires specifying an item distance. The following list describes several linkage methods [HKP11; ML14] and its properties. We denote the incorporated clusters C_i and C_j, respectively. The symbols a and b describe the respective data

item.

single linkage Single linkage calculates the distance between two clusters as the distance of the two closest cluster items.

$$distance_{\text{single}}(C_i, C_j) := \min_{a \in C_i, b \in C_j} \|a - b\|$$

This generally results in clusters in the form of chains.

complete linkage Complete linkage is the opposite of single linkage and uses the two farthermost data items of the two clusters to compute the distance.

$$distance_{\text{complete}}(C_i, C_j) := \max_{a \in C_i, b \in C_j} \|a - b\|$$

This generally results in more circular and compact clusters.

average linkage Average linkage uses the arithmetic mean of the pairwise distances of two clusters as cluster distance.

$$distance_{\text{average}}(C_i, C_j) := \frac{1}{|C_i| \cdot |C_j|} \sum_{a \in C_i, b \in C_j} \|a - b\|$$

The results are rather generic clusters that can have various shapes in-between the outputs of single linkage and average linkage.

centroid linkage Centroid linkage uses the centroids c_i and c_j of the cluster items of C_i and C_j, respectively, to calculate the cluster distance.

$$distance_{\text{centroid}}(C_i, C_j) := \|c_i - c_j\|$$

This linkage also leads to rather generic clusters similar to average linkage, but does not consider the cardinality of the clusters.

Ward's linkage Ward's linkage utilizes the centroid distance, which reflects the distance of the arithmetic means of the data items in the two clusters. By using the normalized squared distance, the distance measures the increase in variance.

$$distance_{\text{ward}}(C_i, C_j) := \frac{distance_{\text{centroid}}(C_i, C_j)}{\frac{1}{|C_i|} + \frac{1}{|C_j|}}$$

This method generally leads to similarly large clusters.

In order to save runtime on subsequent distance computations after merges, it can be useful to update cluster distances instead of recomputing them. The Lance-Williams formula [ML14] can be used to do such updates for various types of linkage.

$$\begin{aligned} distance_{\text{lance-williams}}((C_h \cup C_i), C_j) \coloneqq\ & \alpha_1 \cdot distance(C_h, C_j) + \alpha_2 \cdot distance(C_i, C_j) \\ & + \ \beta \cdot distance(C_h, C_i) \\ & + \ \gamma \cdot |distance(C_h, C_j) - distance(C_i, C_j)| \end{aligned}$$

Each type specifies the parameters α_1, α_2, β and γ. For instance, single linkage can be defined by setting $\alpha_1 \coloneqq 1/2$, $\alpha_2 \coloneqq 1/2$, $\beta \coloneqq 0$ and $\gamma \coloneqq -1/2$.

For clustering large amounts of vectorial data and lowering the runtime costs, BIRCH [ZRL96] is a candidate that combines tree-based indexing and clustering. It uses a cluster feature (CF) tree that is height-balanced and built similarly to a B^+-tree [GWU99]. Cluster features are descriptors of a set of multidimensional data items with the fields N for the number of data items, LS for the sum of data items and SS for the squared sum of data items. By using this data structure, BIRCH can calculate the radius of clusters. BIRCH requires the number of output clusters k as input. In a first step, BIRCH builds the tree and splits cluster features that exceed a certain cluster radius threshold. In a second step, BIRCH groups similar leaf nodes and removes outliers. In a third step, BIRCH applies a clustering algorithm to each leaf node. Experiments show that the runtime of building the tree is nearly linear and the cost of clustering the leaf nodes is determined by the choice of clustering algorithm, e.g. agglomerative clustering. Furthermore, it showed performance weaknesses if the clusters in the data are not spherical in shape [HKP11].

3.4.4. Choice of the Number of Clusters

Since algorithms such as k-means require specifying the number of clusters beforehand, it is an important choice with respect to the quality of clustering results [HKP11]. Generally, the number of clusters is specific for each data set. Experts may be able to estimate suitable values for certain domains and with previous knowledge. In the other cases, finding the right number of clusters is a problem itself.

The Rand index [HA85] is a method that helps to assess the differences in two clusterings [VCH10]. It measures the changes in cluster assignments by looking at the

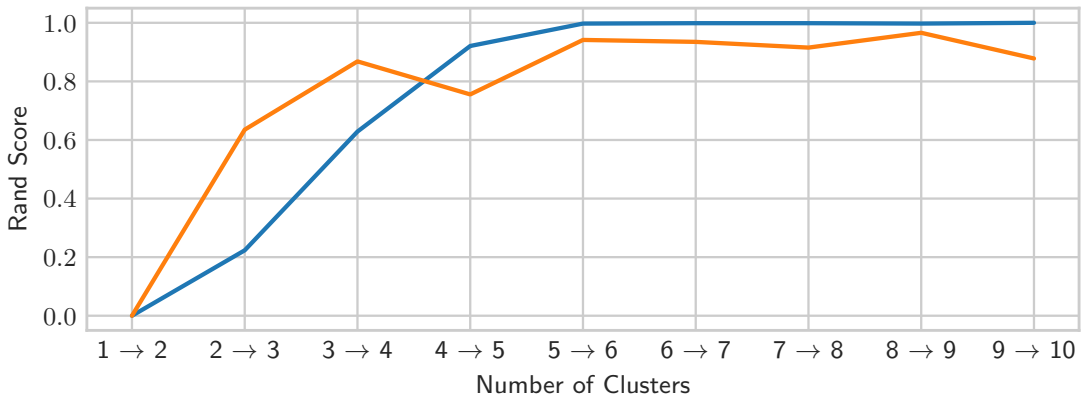


Figure 3.10.: This figure depicts the Rand index of a cluster algorithm on two data sets (blue and orange) in the range of one to ten clusters.

pairwise cluster memberships of the n data items. The formula

$$rand(X, Y) = \frac{a + b}{\binom{n}{2}}$$

calculates the ratio of the number of item pairs a that are in the same cluster in clustering X as well as in Y, the number of item pairs b that are in a different cluster in X as well as in Y and the number of total item pairs $\binom{n}{2}$. If the Rand index is drawn as a line diagram for several numbers of clusters, the function usually presents an elbow where it flattens out. This elbow point is then a sweet spot for choosing the number of clusters. Figure 3.10 illustrates this method by plotting the Rand score of a clustering algorithm for two different data sets. For the blue data set, we see the elbow clearly on the transition from five to six clusters. On the other hand, the orange data set has two potential spots from three to four and five to six, and a notch in between.

The drawback of this approach and similar methods is that it requires computing multiple clusterings. Not only that the output of most clusterings is discarded for the result then, but also the clustering computation is generally very expensive in terms of runtime. Furthermore, it is unclear at which number of clusters to start and in which step sizes to proceed. For most cases, it is infeasible to test all possible parameter settings.

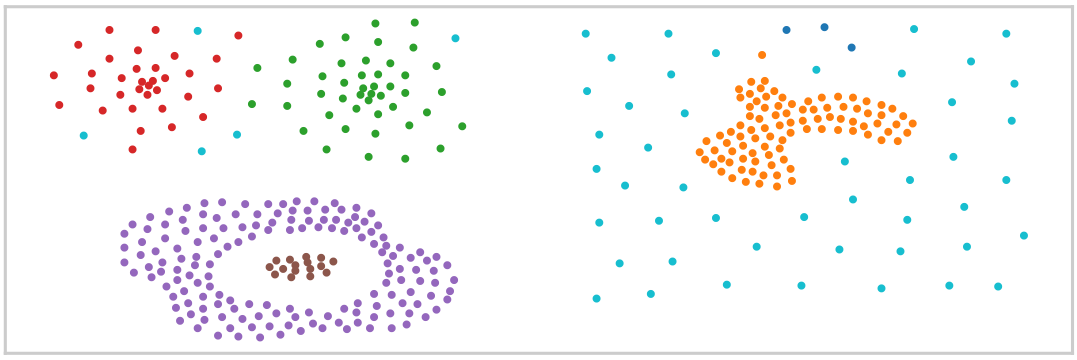


Figure 3.11.: This figure shows a density-based clustering of the *compound* data set [Zah70].

3.4.5. Density-based Clustering

The idea of density-based clustering is to cluster data that is very close to each other in a region of the data domain. The method partitions the data into dense regions and outliers. In contrast to the previous clustering methods, it is necessary to specify a minimum density and not a number of clusters.

The most prominent algorithm in this category is *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) [Est+96]. It requires two parameters ϵ and $minPts$ which specify a radius and a minimal number of points that are required to form a cluster. The algorithm iterates over the data items and assigns each item to one of three classes. Data items are *core points* if they have at least $minPts$ neighbors within their ϵ radius. Data items are *border points* if they are within the ϵ radius of a core point. The algorithm considers all other data items to be *outliers*. These properties require the notion of connectivity or reachability. All items within the ϵ radius are *directly density reachable*. *Density reachability* specifies that two items are reachable from a chain of intermediate items that are pairwise directly density reachable. Border points cannot density-reach another item. This relation is asymmetric. *Density connected* specifies that two items are both density reachable from another intermediate item. Within a cluster all data items are density connected. If a point is density reachable from any point in the cluster, it is also part of this cluster. Figure 3.11 shows the result of a density-based clustering of a data set that has clusters of different forms and densities. In this case, a good choice of the parameters allows extracting almost all clusters with only a few outliers or noise.

Since the algorithm has to perform a radius query (i.e. to calculate distances to all

other items) for each visit of a data item, its runtime is $O(n^2)$ in general. Gunawan [Gun13] was able to show an $O(n \cdot log(n))$ variant for two-dimensional data. In addition, Gan and Tao [GT15] presented an approximate DBSCAN variant that runs in linear time. It produces DBSCAN results in between two settings of ϵ. However, experiments show that this algorithm only works for low-dimensional data in practice [MAS16]. This is due to an exponential increase in runtime in the number of data dimensions. Mai et al. [MAS16] furthermore show pruning strategies to lower the number of radius queries. This leads to a lower runtime in real-world scenarios.

3.5. Quality Measures for Clustering

The evaluation of different clustering methods essentially considers two main criteria. First, the algorithm should be fast since a slow runtime makes it impractical for large real-world use cases. Second, the algorithm has to provide clusters that have a good quality. While the runtime is easily assessable both theoretically and practically, the quality of a clustering is not obviously assessable, in particular when evaluating it mathematically.

Before presenting several quality measures, we have to introduce two categories: intrinsic and extrinsic methods [HKP11]. Extrinsic methods try to work with ground truth data by using human evaluations or comparing it to other clustering results and therefore lead to relative measuring. Intrinsic methods try to find a score without ground truth data by assessing the clustering structure. For instance, they assess how well clusters are separated and how compact clusters are.

3.5.1. Extrinsic Methods

Clustering is usually considered to be an unsupervised learning problem [HKP11] with no obvious ground truth data. However, it is still sometimes possible to receive such ground truth anyway. This is usually done by incorporating expert knowledge that leads to a partitioning of the data. Then it is possible to compare the clustering results to such ground truth data.

There are *matching-based* methods like the *purity* [HKP11]. Purity calculates the extent to which one cluster only contains points from one ground truth parti-

tion.

$$purity \coloneqq \frac{1}{n} \sum_{i=1}^{k} \max_{j=1}^{r} n_{ij}$$

Here, r is the number of partitions, n is the total number of data items and n_{ij} denotes the shared number of items between clustering i and partition j. A value of 1 indicates a perfect purity. The values of k (number of clusters) should be close or equal to r. If every data item is put into a separate cluster ($k = n$), then the purity is automatically 1 as well. Other matching-based methods can be borrowed from evaluating classification algorithms, e.g. *precision* and *recall* [Pow11].

$$precision_i \coloneqq \frac{\max_{j=1}^{r} n_{ij}}{n_i} \qquad recall_i \coloneqq \max_{j=1}^{r} \frac{n_{ij}}{|T_j|}$$

Precision defines the maximum fraction of points from a cluster C_i that is found in a partition. Recall defines the fraction of points in a partition T_j that is common with a cluster. The F_1 score combines precision and recall by calculating their harmonic mean.

$$F_1 \coloneqq \frac{1}{k} \sum_{i=1}^{k} \left(\frac{precision_i^{-1} + recall_i^{-1}}{2} \right)^{-1} = \frac{1}{k} \sum_{i=1}^{k} 2 \cdot \frac{precision_i \cdot recall_i}{precision_i + recall_i}$$

Pairwise methods include, for instance, the already defined Rand index, which can also be used to reflect differences between a clustering result and a partitioning. The *Jaccard coefficient* [HKP11] can be calculated by first calculating the number of data item pairs (true positives / TP) that are in the same cluster and in the same partition.

$$TP \coloneqq |\{(x_i, x_j) \mid y_i = y_j \wedge \hat{y}_i = \hat{y}_j\}|$$

Here, we denote the data items as x_i and label the items in the partitions with y_i and in the clustering with $\hat{y}_i$. The true negatives (TN) are calculated similarly by calculating the number of data items that are neither in the same cluster nor in the same partition.

$$TN \coloneqq |\{(x_i, x_j) \mid y_i \neq y_j \wedge \hat{y}_i \neq \hat{y}_j\}|$$

Finally, the coefficient divides the true positives by the total number of data item pairs n, but ignores the true negatives.

$$Jaccard \coloneqq \frac{TP}{\binom{n}{2} - TN}$$

3.5.2. Intrinsic Methods

Intrinsic methods assume that there is no ground truth for clustering results. Thus, the idea of measuring the quality is to assess the previously mentioned intra cluster similarities and inter cluster distances [HKP11]. Note that it is not evident that the data items really reflect the distribution of the real-life scenario they were extracted from (e.g. via observations). Furthermore, the choice of distance function is of utmost importance because it changes the relation of data items and, thus, the quality assessment. As discussed previously, the number of clusters is unclear as well, but can be important for the quality of the clustering result. The following quality measures try to express quality with different methods.

The *silhouette coefficient* [HKP11] is an intrinsic measure that compares the average distance of a data item x of cluster C_i to items within the same cluster with the average distance to items of different clusters.

$$silhouette(x) = \frac{\min_{1 \leq j \leq k, j \neq i} \left\{ \frac{\sum_{x' \in C_j} distance(x,x')}{|C_j|} \right\} - \frac{\sum_{x' \in C_i, x' \neq x} distance(x,x')}{|C_i|-1}}{\max \left\{ \min_{1 \leq j \leq k, j \neq i} \left\{ \frac{\sum_{x' \in C_j} distance(x,x')}{|C_j|} \right\}, \frac{\sum_{x' \in C_i, x' \neq x} distance(x,x')}{|C_i|-1} \right\}}$$

Data items that have a silhouette coefficient close to 1 are considered to be well clustered, items with a low coefficient may be considered as outliers.

The *Dunn index* (DI) [VCH10] calculates the fraction of the minimum distance between clusters (δ_{ij}) and the maximum intra cluster distance between data items (Δ_i).

$$\delta_{ij} := \min_{x \in C_i, x' \in C_j} distance(x, x')$$

$$\Delta_i := \max_{x, x' \in C_i} distance(x, x')$$

$$DI := \frac{\min_{1 \leq i < j \leq k} \delta_{ij}}{\max_{1 \leq i \leq k} \Delta_i}$$

The *Davies-Bouldin index* [VCH10] calculates the scatter for each cluster and tries to find in each case another cluster that leads to the maximum fraction of the sum of their scatter values and a value that measure their separability. The scatter is calculated by computing the distances of the data items of a cluster to its centroid. The separation to the other clusters is calculated by using the distances between their centroids. The final index is calculated by averaging over the scores (fractions) of each individual cluster.

4

Quality Measures for Visual Point Clustering

This chapter deals with the quality assessment of visual point clustering results. First, Section 4.1, gives a motivation why it is necessary to introduce a new set of quality measures. Then, Section 4.2 introduces preliminary mathematical definitions and notations. This helps to achieve a uniform understanding of the concepts and to make the following definitions more compact. Section 4.3 discusses the assignment of the visualized results to their corresponding input data. This is a necessary setup step in order to apply the quality measures to any algorithmic output. Section 4.4 presents the core of the chapter, which is the definition of seven quality measures. Each measure is informally motivated and complemented by a precise mathematical definition. Section 4.5 discusses the problem of mapping a generic clustering result to a set of circles. Here, several definitions form rules for this mapping, which are later evaluated in the experiment section. In Section 4.6, we present experimental results of a set of clustering methods with respect to visual point clustering. This section evaluates different parameter settings for each type of clustering method against multiple data sets. Furthermore, we measure and discuss the impact of circle mappings on the overall result. Subsequently, Section 4.7 presents an extension of the quality measures for the visual point clustering of data containing multiple classes. Here, we develop the necessary adaptations for each individual measure to cope with multiple classes and discuss possible implications to the measure results. Finally, Section 4.8 provides a brief summary of this chapter.

4.1. Motivation

In Chapter 2, we defined the specific problem of visual point clustering, i.e., clustering a set of points and representing these clusters as circles on a map. Furthermore, we briefly discussed aspects that are important in visual point clustering. These

range from avoidance of overlaps or obscuring information to the representation of information about the underlying data. In order to compare the clustering methods, it is reasonable to develop the concept of quality on a formal foundation rather than a human intuition of the results. In particular, this is advantageous if we want to assess existing clustering methods or to develop new ones in order to tackle the visual point clustering problem.

In Section 3.5, we looked at various clustering measures that are often used to evaluate clustering results for different data sets. However, it is not possible to reasonably apply these to the visual point clustering problem setting. This is due to the differences in what we call *measurement of intention* vs. *measurement of perception*, on which we now elaborate in more detail.

In the generic clustering assessment, it is common to use the output of the algorithm to compute the quality, e.g. the difference between intra and inter cluster distance. We denote this as *measurement of intention* because we directly use the intended cluster labels that were provided by the algorithm for each data item.

On the contrary, in the visual point clustering problem setting we have to consider the user, who is in the end the one that visually extracts the information out of the clustering result. Since we want to have measures that correlate with the user's perception, we call this attempt *measurement of perception*. In order to assess the perception, we have to actually look at the set of circles on the map and deduce from that the connection to the underlying raw data. This connection, in combination with the circles that represent the clusters, is then used as input for the various quality measures.

It is very difficult to derive a single quality measure that evaluates all perceptible desired properties at once. Hence, we will provide a list of important properties and then address them individually by introducing several measures for different aspects. This allows us to evaluate them independently in subsequent experiments. Furthermore, we can create a joint measurement that combines all individual quality measures into a single number. This approach makes it much easier to incorporate different quality aspects and to compare the quality of the output of different algorithms for different data sets.

The experiments will show that each criterion examines different aspects of the selected algorithms. This will allow us to investigate the effects of different parameter settings and to identify potentially conflicting aspects. Overall, the quality measures are practical criteria for evaluating visual point clustering.

4.2. Preliminaries

In this section, we define the essential notation as well as functions that we will use throughout this chapter. These range from transformations of the data space to auxiliary formulas. These formulas ensure that all readers have a common understanding of the definition of quality metrics.

Points and Circles For a point $p \in P$ we use $p.x$ and $p.y$, respectively, to address the x and y coordinate. For a circle $c \in C$ we use the same coordinate notation to define the center of the circle and additionally use $c.r$ to specify the radius of this circle. The coordinates and radii are given in map units. Furthermore, we abbreviate $n \coloneqq |P|$ and $m \coloneqq |C|$.

Map In general, maps can have different x and y extents (cf. Section 3.1). Without loss of generality, we only consider quadratic maps and define the measures only for quadratic maps. As described in Section 2.2, we want to calculate visual point clustering for different zoom levels z. These have been defined as a magnification of the whole map, i.e., scaling it up for larger zoom levels. This leads to the problem that we have to define measures for maps of different sizes. This would inevitably lead to inconsistent maximum distances that we have to cope with. Therefore, we transform all outputs linearly into the unit square $[0, 1] \times [0, 1]$. This allows us to define the quality measures independent of the zoom level. Note that this leads to a reduction of the circle radii with the factor $1/2^z$. Since the mapping is linear, there is no distortion of distances or positioning. As a consequence, the minimum Euclidean distance between any two points or circle centers is 0 and the maximum distance is $\sqrt{1^2 + 1^2} = \sqrt{2}$, respectively.

Circle-Point Distance To measure Euclidean distances between circles and points, we consider only the coordinates of the circle's center and ignore its radius. This leads to:

$$pdist(c, p) \coloneqq \sqrt{(c.x - p.x)^2 + (c.y - p.y)^2}$$

In case we want to address whether a point is inside or outside a circle, there is an additional definition that incorporates the radius:

$$cpdist(c, p) \coloneqq pdist(c, p) - c.r$$

Note that the value is 0 when the point is at the edge of the circle and is negative iff the point is inside the circle.

Circle Distance The distance between two circles is defined as follows:

$$cdist(c_1, c_2) \coloneqq pdist(c_1, c_2) - c_1.r - c_2.r$$

Iff the value is zero or positive, the circles do not intersect each other.

Centroid The centroid specifies the geometrical center of a set of points $P' \subseteq P$.

$$centroid(P') \coloneqq \left(\frac{1}{|P'|} \sum_{p \in P'} p.x, \frac{1}{|P'|} \sum_{p \in P'} p.y \right)$$

Analogously, for a set of circles $C' \subseteq C$, $centroid(C')$ gives the centroid of the coordinates of the circles by ignoring the radius component.

Area of Objects For calculating the area of an arbitrary object o, we use the function $A(o)$. If the object is a circle c, we use the well-known formula $A(c) \coloneqq (c.r)^2 \cdot \pi$. Furthermore, for a point p, the area $A(p)$ is 0. Since the computation of the area of arbitrary objects is very complex, inefficient or lacking an analytical solution, we use Monte Carlo simulations [GS07] to estimate the area.

Intersections and Unions The associative operators $c_i \cap c_j$ and $c_i \cup c_j$ define the geometric intersection and the union of circles c_i and c_j, respectively. In the following, we will use these operators mostly in combination with either a point containment or an area calculation. For the former, we check whether one or all of the points are within the circles. For the latter, we make use of the Monte Carlo simulations. Both cases require the calculation of $cpdist(c_i, p) \leq 0$ that indicates whether a point p is within circle c_i.

4.3. Visual Assignment

Since we want to measure the perception of the circle representation (cf. Section 4.1), it is insufficient to use the output of a clustering algorithm directly to measure the quality in our problem setting. Therefore, it is necessary to have functions that derive cluster assignments from a set of circles C (visual point clustering) and a set of points P (raw data). The task is approached visually as if the user would evaluate a (translucent) circle map on top of the raw points – this would not be possible in practice when thinking of millions of raw data points, but is possible in computational thinking. However, a single assignment function does not cover all aspects that are necessary for defining the quality measures. Hence, we introduce two different assignments that are both valid and useful in different scenarios.

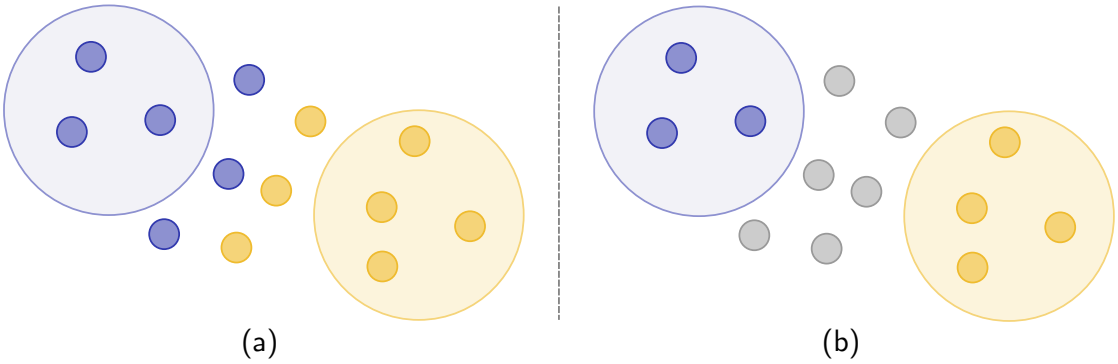


Figure 4.1.: This figure illustrates the differences between the nearest neighbor assignment (a) and the enclosing assignment (b). The blue points are associated to the light blue circle while the yellow points are associated to the light yellow circle. The gray points are not assigned to any circle.

4.3.1. Nearest Neighbor Assignment

The nearest neighbor assignment $\mathcal{N}$ of $c_i \in C$ is defined as

$$\mathcal{N}_i := \{p \in P \mid c_i = \arg\min_{c \in C} \ dist(c, p)\}$$

with

$$dist(c, p) := \begin{cases} cpdist(c, p) & \text{if } cpdist(c, p) > 0 \\ -\sqrt{2} + pdist(c, p) & \text{else} \end{cases}.$$

It assigns the points to the nearest circle. If a point is outside a circle, the assignment uses the distance to the perimeter of the circle. Otherwise, the circle encloses the point and the assignment considers the distance to the center. Since the maximum distance in the unit interval is at maximum $\sqrt{2}$, this inverse calculation results in the smallest (negative) distance iff the circle center and the point are equal. In general, this leads to the natural assignment of a point to the most inner (containing) circle in the case of multiple overlapping circles. Otherwise, the point is assigned to the nearest circle. Figure 4.1 (a) provides an example of this assignment that shows the aforementioned properties. The points are colored blue and yellow and are assigned to their nearest circle.

4.3.2. Enclosing Assignment

The enclosing assignment $\mathcal{E}$ of $c_i \in C$ is defined as

$$\mathcal{E}_i := \{p \in P \mid contains(c_i, p)\}$$

with the predicate $contains(c, p) := cpdist(c, p) \leq 0$. Moreover, there exists a residual assignment $\mathcal{E}_\emptyset = P \setminus (\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_m)$ which contains all points that are not contained in any circle.

This assignment attaches each point to every circle that encloses that point. Note that in contrast to the nearest neighbor assignment, points can be attached to multiple circles in the case of circle overlap. Without any overlapping circles, the mapping is still an *at most one* link between a point and a circle because the residual assignment may not be empty anyway. Figure 4.1 (b) illustrates this assignment for an exemplary set of points and circles. In comparison to the nearest neighbor assignment, it provides the same mapping for the points within the circles (blue and yellow) but also the residual points (gray) that are not assigned.

4.4. Quality Measure Definitions

In the following, we define a set of quality measures M assessing the quality of a circle representation C for a set of points P. The measure $\mu_i \in M$ assesses a requirement i, i.e., one aspect of a qualitative circle representation of the underlying data. Note that the index i is not a number in the following, but a symbolic name with $i \in \{area, centered, overlap, distance, unassigned, uniform, zoom\}$. A measure μ is a utility function (i.e. users prefer inputs that lead to higher values than to lower values) that maps the input to a value of the unit interval $(0, 1]$. This ensures that all measures are within a comparable data range and that the intercomparability of the requirements is guaranteed.

It is difficult, however, to find an intuitive definition of all measures that directly maps to the unit interval. This is especially the case if there are unknown bounds, e.g. in case of an error value. Hence, we measure for each requirement the deviation ν_i of C to the *optimal* solution. Moreover, the output of ν_i must be greater than or equal to zero since it reflects an error. We now apply a transformation from deviations to utilities by using the mapping function $\mu_i := e^{-\nu_i}$. The exponential function ensures that for deviations of zero, the utility value is exactly one. On the other hand, larger values lead to an exponential decrease of the utility towards zero. For the sake of simplicity, we will evaluate each requirement on the basis of the deviation ν_i and omit the implicit transformation into a utility.

As it will turn out later, some ν_i values have an upper limit in addition to the required lower limit, e.g. a lower limit of zero and an upper limit of one. This

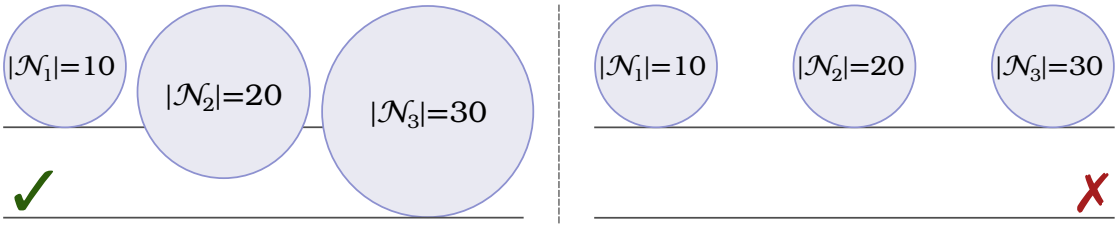


Figure 4.2.: This image shows a good and a bad example for *area proportionality* in juxtaposition. While the left side shows circles whose areas scale linearly in the number of assigned points, the right side shows circles whose sizes do not reflect the number of points.

means that the utility value that applies the deviation as a negative value in the exponential function would be in the interval $[0.36, 1]$. Since this would lead to a loss of range, we apply a linear stretch function to utilize the entire range again.

4.4.1. Area Proportionality

Idea The idea of area proportionality is that the area of the circles should be proportional to the number of associated points. This helps the user to identify the differences in cardinality of clustered data spots (circles) on the map. Hence, we want to penalize circles that do not possess this property.

Definition In order to define this deviation, we first need a definition for the density of a circle. Then, we can measure whether the densities of all circles are almost equal. The density d_i for a circle c_i is given as

$$d_i := \frac{|\mathcal{N}_i|}{A(c_i)} .$$

It uses the nearest neighbor assignment, which reflects the data in the area in and around a circle. To quantify *almost equal*, we use the coefficient of variance [Bon06]

$$\nu_{area} := \frac{\sqrt{Var(\{d_1, \ldots, d_m\})}}{E(\{d_1, \ldots, d_m\})} = \frac{\sqrt{\frac{1}{m}\sum_i^m (d_i - \frac{1}{m}\sum_i^m d_i)^2}}{\frac{1}{m}\sum_i^m d_i} .$$

This measure obviously penalizes strong variations in the d_i values, which is the desired property of the measure. With regard to the robustness of the

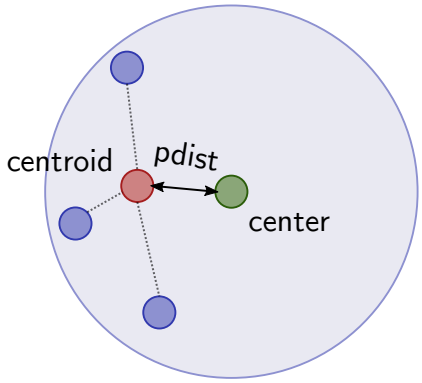


Figure 4.3.: This figure illustrates the calculation of the *circle points centered* quality measure. It calculates the point distance between the center (green) of the circle and the centroid (red) of the points (blue) that are associated to the circle.

computation, one can also use the quartile coefficient of dispersion [Bon06]

$$\nu_{area'} := \frac{Q_3 - Q_1}{Q_3 + Q_1} ,$$

where Q_i is the i-th quartile of all density values. It uses the first and third quartiles for calculating the difference (interquartile range [HKP11]) and for normalization. While the first definition indicates the error in relative density units and has no upper bound, the second definition has an upper limit of 1 since all density values are positive.

Example Figure 4.2 shows an exemplary evaluation for this measure for a small set of three circles. The values in the circles indicate the cardinality of the assigned points. While the circles on the left side facilitate estimating the differences in cardinality due to their different areas ($\nu_{area} = \frac{Var(\{0.1,0.1,0.1\})}{E(\{0.1,0.1,0.1\})} = 0$, $\nu_{area'} = \frac{0.01-0.01}{0.01+0.01} = 0$), the right side has no such indication and is thus misleading ($\nu_{area} = \frac{Var(\{0.1,0.2,0.31\})}{E(\{0.1,0.2,0.3\})} = 48.265$, $\nu_{area'} = \frac{0.03-0.01}{0.03+0.01} = 0.5$). The figure clearly shows that this property benefits the user and should be considered.

4.4.2. Circle Points Centered

Idea The idea for centered points in a circle is that the centroid of the points assigned to a circle should be close to the center of the circle. Otherwise, the circle would somewhat obliquely cover the points. It would be particularly

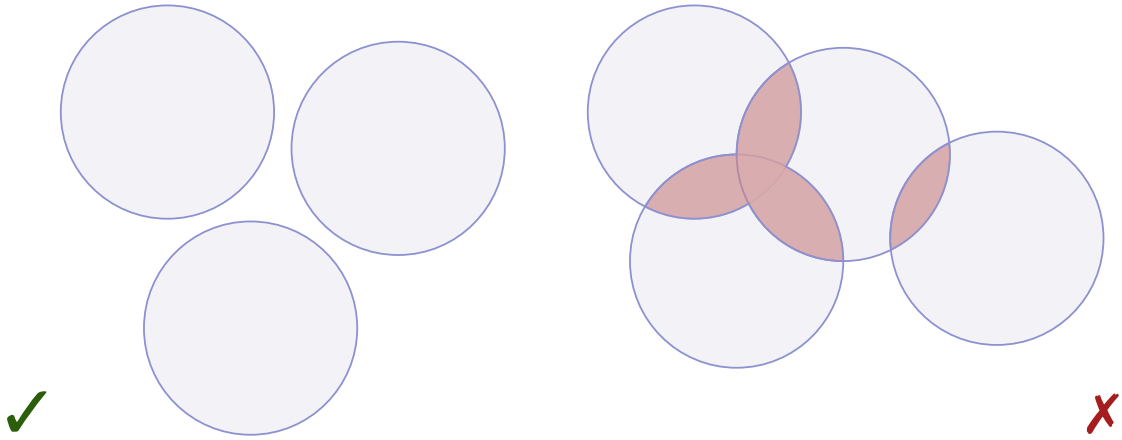


Figure 4.4.: This figure presents a good example (left) and a bad example (right) for the *circle overlap* quality measure. The red colored area shows the overlapping of circles, which covers parts of the other circles, and thus reduces the information value. Each overlap reduces the quality score.

unexpected for the user to assume that the centroid of the points attached to a circle did not lead to the centroid of that circle.

Definition For each circle, we compute the distance between its center and the centroid of the points assigned to the circle. Then, the m distances are averaged to obtain a single deviation value. This yields the following measure:

$$\nu_{centered} \coloneqq \frac{1}{m} \sum_{i=1}^{m} pdist(c_i, centroid(\mathcal{N}_i)) \ .$$

Example Figure 4.3 provides a visual example of $\nu_{centered}$. Here we get a sense of the impact of such obliqueness of circles with respect to the data points. Since the blue points are only at the left of the light blue circle, the center of the circle and the centroid of the points are clearly separated from each other. For a user, it is hard to estimate the underlying distribution this way without seeing the points. Shifting the circle to the left (by the value of the *pdist* calculation) improves the visualization and the quality score.

4.4.3. Circle Overlap

Idea The *circle overlap* measure addresses the requirement that the optimal solution does not contain overlaps, and thus no occlusion. Chapter 2 has already addressed this fundamental property with regard to the problem formulation,

and Chapter 1 provided an example why this key problem leads to undesirable results. Since the avoidance of occlusion is a reason for applying visual point clustering to the problem, any residual occlusion that is an indicator of hidden information must be penalized by this deviation value.

Definition The *circle overlap* deviation must directly reflect the degree of the overlap of the circles. It is not necessary to include any points into this calculation. The optimum is no overlap at all. We therefore measure the deviation from the optimum by

$$\nu_{overlap} \coloneqq \frac{A(C_1 \cap \ldots \cap C_m)}{A(C_1 \cup \ldots \cup C_m)} ,$$

which is to some extent a modification of the well-known Jaccard coefficient. In the worst case, if everything overlaps to 100 %, this leads to a deviation of $\nu_{overlap} = 1$. We then have to apply the stretching function for the utility value.

Example Figure 4.4 depicts an example of not or partially overlapping circles to illustrate this measure. Here, the three circles on the left side depict a perfect score. The four circles on the right side, in contrast, show a substantial overlap, and thus occlusion of information in the visualization. While the worst case would be a complete overlap of all circles, this example poses a diminished quality score.

4.4.4. Circle Point Distance

Idea The *circle point distance* measure reflects the fact that users are unable to identify whether a circle represents (hidden) points that are far away from the circle as a representative. Therefore, points (far) outside their representative circle are unintentional, and thus indicate a bad quality of the circle representation. Ideally, all points are enclosed by the circle. Otherwise, the further away the points are located from a circle with the perspective of the nearest neighbor, the greater the deviation value should be.

Definition In order to achieve this deviation value, we first measure how far the points are outside their representing circle. We use the nearest neighbor assignment here. Then, we compute all values for each circle and aggregate them in order to obtain a single value. Concretely this is done by

$$\nu_{distance} \coloneqq \frac{1}{m} \sum_{c \in C} \sum_{p \in \mathcal{N}_c} \mathcal{I}_{c,p} \cdot cpdist(c,p) .$$

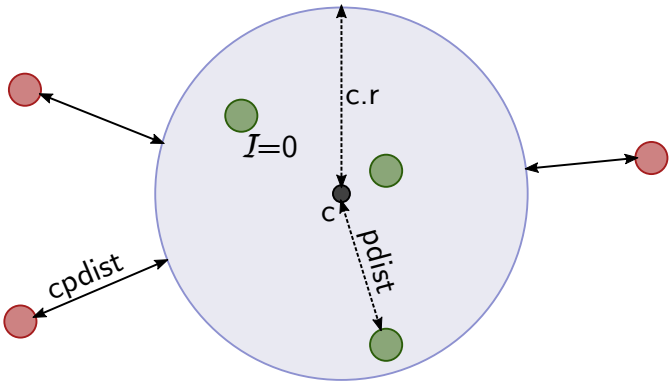


Figure 4.5.: This figure depicts the calculation of the *circle point distance* quality measure. The green points do not represent an error for the measure. On the other hand, the red points reduce the quality score by the distance from their representing circle (*cpdist*).

The indicator function is defined as follows:

$$\mathcal{I}_{c,p} := \begin{cases} 1, & \text{if } cpdist(c,p) > 0 \\ 0, & \text{otherwise} \end{cases} .$$

To evaluate $\nu_{distance}$ we only incorporate points that are not enclosed by their representative circle. All points that are within their representative circle produce a negative value (distance) that would distort the result. The incorporation of the indicator function $\mathcal{I}_{c,p}$ prevents this.

Example In Figure 4.5, we see the distances (*cpdist*) of points (red) that are outside of their representative circle. We have to consider that the distances are exponentially factored in the utility value due to the transformation from the deviation value. Therefore, we can envision this as Gaussian distributions that start at the center of each circle which are modified to have a value of one within the circles. In this example, the green points have an indicator function value of $\mathcal{I} = 0$ so that they do not affect the quality score.

4.4.5. Unassigned Points

Idea *Unassigned Points* addresses the circumstance that it is undesirable to have points that are not contained in any circle. A user would see an uncovered area that actually contains data. This leads to a biased reception of the map. Hence, the more points a solution C does not cover, the more C deviates from an optimal solution.

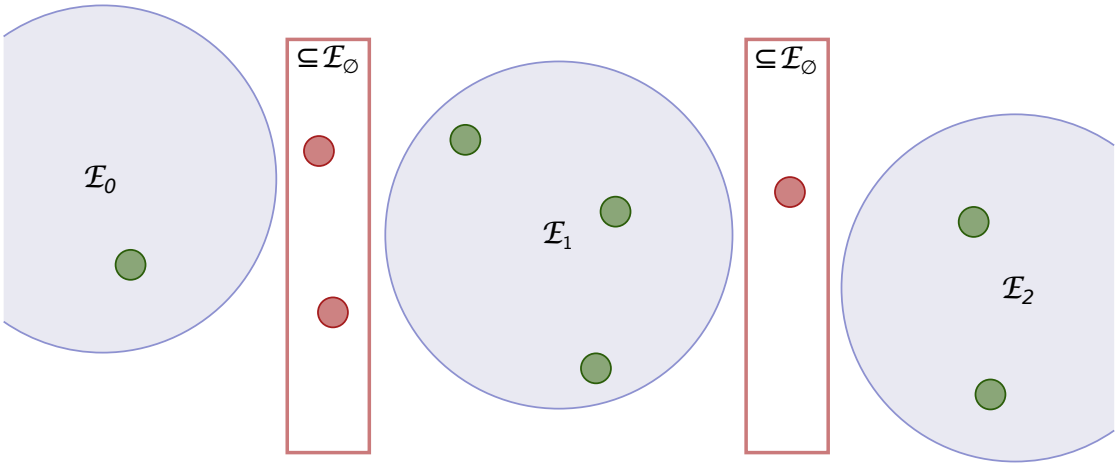


Figure 4.6.: This figure illustrates the *unassigned points* quality measure and the connection to the enclosing assignment. The red boxes represent subsets of the points (red) in the residual assignment. In contrast to the green points, which do not pose an error, they reduce the quality score by their cardinality.

Definition We assess this concept with the following measure:

$$\nu_{unassigned} \coloneqq \frac{|\mathcal{E}_\emptyset|}{n} \, .$$

This formula uses the enclosing assignment and exploits the residual assignment $\mathcal{E}_\emptyset$, which is the set of points that is not enclosed by any circle. The deviation value calculates the fraction of points in the residual assignment and the total number of points n. In the optimal case, $|\mathcal{E}_\emptyset| = 0$ and in the worst case, $|\mathcal{E}_\emptyset| = n$. This leads to a lower bound of zero and an upper bound of one. Therefore, we again need to apply the stretching function to utilize the entire range of the unit interval after applying the exponential function to the negative deviation.

Example Figure 4.6 presents an example that visually shows the computation of this measure. The measure ignores all points that are enclosed by a circle (green points of $\mathcal{E}_1, \ldots, \mathcal{E}_3$). The red boxes highlight the subsets of the residual assignment $\mathcal{E}_\emptyset$. These outside points (red) are all treated equally and the exponential scaling of the utility value leads to a rapid punishment of uncovered points. Therefore, a significant number of uncovered points leads to a utility towards zero.

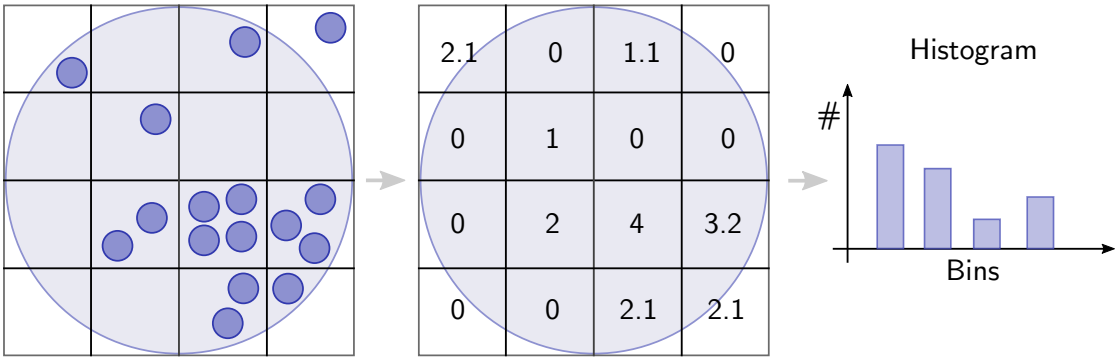


Figure 4.7.: This figure illustrates the histogram computation of the *uniform point distribution* quality measure as a three step process. The first step shows a grid and the binning of points into grid cells. The second step illustrates the bin counts and the scaling of the outer bins that only partially intersect with the circle. The third step shows the creation of a histogram from the bins in order to assess the uniformity of the point distribution within the circle.

4.4.6. Uniform Point Distribution

Idea The optimal distribution of points within their representing circle should be uniform. A user cannot expect to have, for instance, two dense, opposite point hot spots rather than a homogeneous distribution of the points. This means that it is not desirable to deviate strongly from a uniform distribution for points that are represented by a circle. Ideally, the points that are enclosed by their representative circle should follow a uniform distribution. In the worst case, the points in a circle are heavily skewed.

Definition We measure the deviation of this optimal distribution as follows: We partition a circle c_i into a two-dimensional grid of $\left\lceil \sqrt{|\mathcal{E}_i|} \right\rceil$ square buckets by applying the function $histogram(c_i, \mathcal{E}_i)$. This follows the *square-root choice* rule [Sco15]. For histograms, a good choice of buckets is essential in order to obtain a valid representation of the distribution. The above-mentioned choice has been empirically proved to be suitable in this scenario.

For each bucket, we count the number of points that fall into that bucket. Our counting also takes the case into account that a bucket is only partially enclosed by a circle. In this case, we interpolate the count by the fraction of its contained part. This can be done by calculating the ratio of the area of the circle-square intersection to the area of the square. Since an analytical solution is rather complex, we fall back to our Monte-Carlo-based area

function from Section 4.2.

Similar to the ν_{area} measure, we employ the coefficient of variance to measure the deviations of the individual bucket counts. Equal counts in all buckets correspond to a uniform distribution. Hence, the coefficient of variance is an appropriate measure to check this property. We aggregate the results for individual circles by taking the average of the coefficients of variance and define $\nu_{uniform}$ as

$$\nu_{uniform} := \frac{1}{m} \sum_{c_i \in C} \frac{\sqrt{Var(histogram(c_i, \mathcal{E}_i))}}{E(histogram(c_i, \mathcal{E}_i))} .$$

Note that $histogram(c_i, \mathcal{E}_i)$ outputs a set of bucket sizes (integers). Alternatively, a Kolmogorov-Smirnov-Test [LRH07] allows measuring the deviation as a confidence value of uniformity. However, the spatial histogram proved to be computationally more robust for particularly smaller point sets within a circle. In any case, we employ the enclosing assignment here, since we are only interested in the points within the circular areas.

Example Figure 4.7 illustrates the histogram creation within the boundaries of a single circle in three steps. At first, the measure overlays the circle with a grid. For each grid cell, the measure then counts the number of contained points. The second step emphasizes the scaling of border grid cells that only partially intersect with the circle. As these cells represent a smaller area of the circle, they scale the counted value up by the fraction of cell area and circle-cell intersection area. Thus, e.g. the left upper cell represents a value of 2.1 even though there was only one point falling into it. In contrast, the upper right cell emphasizes that points are not considered if they are outside the circle (but within the cell's bounds). Furthermore, the third step shows the retrieval of uniformity when drawing the histogram of the cell count values. Optimally, all histogram bins have an equal height (= number of elements).

4.4.7. Zoom Consistency

Idea Zooming into an area should break up clusters and give a more detailed view on the point distribution. This means, we expect that smaller circles are located in the regions that are covered by bigger circles from the previous zoom level. Users should not zoom into an area that is covered by a circle and find nothing there. In this case, the user cannot be sure whether the view before or after zooming is correct or erroneous. This notion of zoom

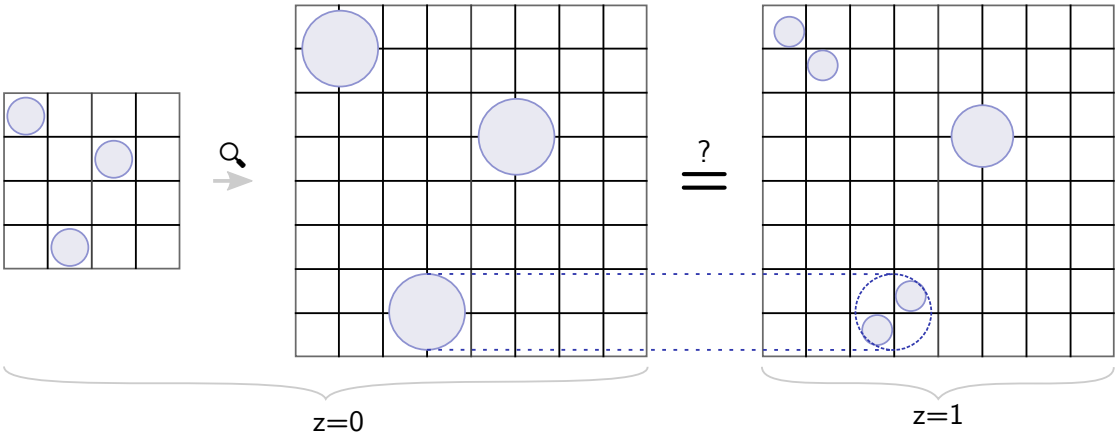


Figure 4.8.: This figure shows an example of the *zoom consistency* quality measure. It illustrates the comparison of two subsequent zoom levels and the calculation of the intersection (dotted line and circle). The arrow with the magnifying glass depicts the (theoretical) up-scaling of the lower zoom level in order to facilitate the comparison. The grid in the background only serves the purpose of comparability between the sets of circles.

consistency is stricter than the ones from Sarma et al. [Das+12] and Guo et al. [Guo+18]. They only require the objects to be in the magnified area. We furthermore require them to be at the correct spot.

Definition Since this measure is the only one that requires two sets of circles as input (in contrast to a set of points and a set of circles), we have to slightly extend our definitions. We define the set of circles at zoom level z as $C^{(z)}$. To check the zoom consistency property for an individual circle $c \in C^{(z)}$ we measure the relative overlap with the next zoom level $z+1$. To do this, we calculate the intersection of a circle c with the combination (union) of the circle from the next zoom level and divide it by the area of that circle. Finally, we aggregate over all circles in $C^{(z)}$ and calculate the average. This leads to

$$\nu_{zoom} := \frac{1}{m} \sum_{c \in C^{(z)}} \frac{A\left(\left(\bigcup_{c' \in C^{(z+1)}} c'\right) \cap c\right)}{A(c)} .$$

One could argue that we have to address the differently sized maps of zoom level z and $z+1$. However, due to the scaling to the unit interval mentioned in Section 4.2, it is not necessary to adjust the sizes between contiguous (or arbitrary) zoom levels. Despite the complex, nested union and intersection

computations, the definition of our area computation allows us to easily express this with Monte Carlo simulations.

Example Figure 4.8 provides an example of how the quality measure assesses two contiguous zoom levels to calculate the deviation ν_{zoom}. We depict the initial scaling of any input to the unit interval. The arrow with the magnifying glass indicates this on the left side of the illustration. Furthermore, we provide a background grid to facilitate the perception of the breakup of clusters when zooming in ($z = 1$). Obviously, the circle shape does not allow perfect scores when breaking up circles. This is comparable to the circle packing problem [Can+07] where the goal is to reduce the dead space. The dotted two lines and the circle indicate this for one instance on the lower part of the figure.

4.5. Clustering Circle Mapping

As described in Chapter 3 and discussed in Section 4.1, general clustering algorithms output a mapping of points to clusters. However, for visual point clustering, the output must be a set of circles C that represents such points P. In this section, we close the gap and define transformation functions. They are in particular used and evaluated in the experiments in Section 4.6.

Let $G_i \coloneqq \{p \in P \mid p \text{ is assigned to Cluster } i\}$ be the i^{th} assignment. We then apply a transformation function τ to each cluster, which maps it to a circle. This provides the mapping to a visual point clustering. It is, however, not trivial to define such a mapping τ that is universally valid and meaningful. We therefore define three methods that are candidates for τ. The experiment section (cf. Section 4.6) will evaluate the quality of such candidates.

4.5.1. Circumcircle

A trivial method for mapping a set of points to a circle is to use a bounding object. In our scenario, this is the boundary circle. Hence, this mapping function uses the centroid of the assigned points for calculating the center of the circle. To define the radius, we use the farthermost point to the center. Thus, we obtain

$$\tau_{cc}(G_i) \coloneqq \begin{pmatrix} centroid(G_i) \\ \max\left(\max_{p' \in G_i} \left(pdist\left(centroid(G_i), p'\right)\right), r_{min}\right) \end{pmatrix} \begin{matrix} \text{coordinates } x \text{ and } y \\ \text{radius} \end{matrix}.$$

We expect this mapping to produce quite large circles. It should be particularly sensitive regarding outliers, which would drastically increase the diameter. Besides that, the outermost maximum function ensures that the circle radius is never smaller than the required minimum radius r_{min}. We discussed the concept and visual implications of it in Chapter 2.

4.5.2. Log_2

Jänicke et al. [JHS13] presented a mapping function that uses the cluster centroid and scales the radius with regard to the number of assigned points. While the centroid definition is identical to the one in τ_{cc}, the specification of the radius uses a completely different approach. They used an empirical maximum circle area

$$A_{max} \coloneqq (4 \log_2 (n+1))^2 \cdot \pi$$

and a minimum circle area

$$A_{min} \coloneqq r_{min}^2 \cdot \pi$$

with a fixed r_{min}. Using these two values leads finally to

$$\tau_{log_2}(G_i) \coloneqq \begin{pmatrix} centroid(G_i) & \text{coordinates } x \text{ and } y \\ \sqrt{\frac{A_{min} + \frac{|G_i|-1}{n-1}(A_{max} - A_{min})}{\pi}} & \text{radius} \end{pmatrix}^T .$$

The radius will grow proportionally in the number of points between the minimum and maximum area. Furthermore, the scaling is implicitly bounded by the minimum radius.

4.5.3. Log_{10}

Since the results of Jänicke et al. were empirical, we add another maximum area function that uses the decimal logarithm. This implies a doubling of the size by an increase of an order of magnitude of points. We use the identical definition of the minimum area and further define

$$A_{max} \coloneqq (\log_{10}(n) \cdot r_{min})^2 \cdot \pi \ .$$

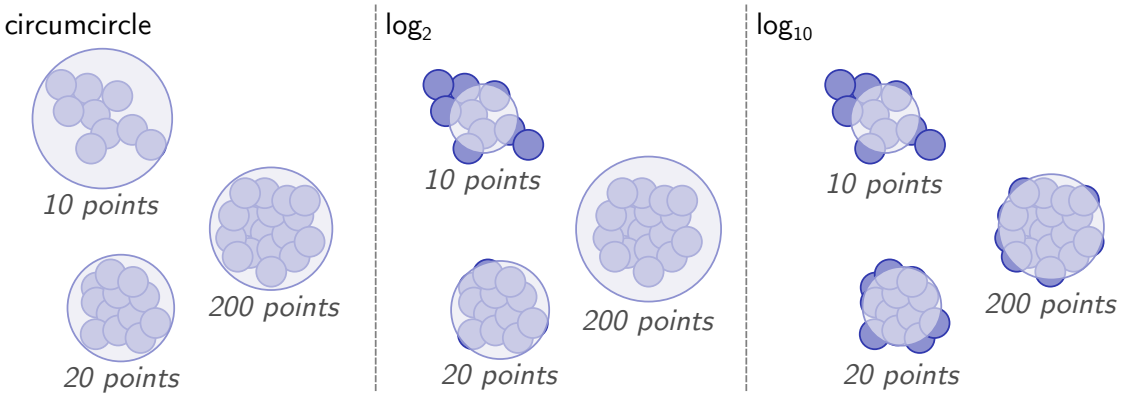


Figure 4.9.: This figure shows the differences of the three transformation functions *circumcircle*, log_2 and log_{10} for the same set of clusters.

We still have the same transformation function (for the sake of completeness):

$$\tau_{log_{10}}(G_i) := \begin{pmatrix} centroid(G_i) \\ \sqrt{\frac{A_{min} + \frac{|G_i|-1}{n-1}(A_{max} - A_{min})}{\pi}} \end{pmatrix}^T \begin{matrix} \text{coordinates } x \text{ and } y \\ \\ \text{radius} \end{matrix} .$$

The scaling here can be desirable when dealing with larger data sets, as the growth is smoother or less drastic. In addition, the maximum area is smaller. This can make sense because of limited map space. Note, however, that the maximum area is only used if there is only one circle representing all points.

In Figure 4.9, we depict three exemplary visual point clusterings of the same set of points by using the three proposed transformation functions. We see, for instance, that *circumcircle* leads to no points that are outside a representative circle. On the other hand, depending on the point distribution in a cluster, the area of the circles may not indicate the cardinality. This is, in contrast, the case with both log_2 and log_{10}. Here we see the differences in the increase of the circle area as the number of points increases. With regard to the cardinality of the input data set, one or the other can lead to a better fitting visualization.

4.6. Experiments

In this section we investigate how existing clustering methods work in the visual point clustering problem setting. For this, we describe the methods and their implementation and state the parameters which are required to be set by the user.

Since the optimal parametrization for the methods is unclear, we propose to investigate a range of parameters for each method. Within these ranges, we perform a grid search in order to find the best possible setting. In addition, this allows us to examine the impact of different parametrizations. This also reveals the applicability of the methods and their suitability of determining such parametrizations in a universal scenario for arbitrary data sets.

Furthermore, we investigate the influence of the transformation function τ on the quality of visual point clustering. We combine each previously defined (cf. Section 4.5) transformation function with each of the clustering methods. As a result, there are many combinations that allow a fair investigation of the impact of such functions. Altogether, the parameters of the clustering methods and the choice of the transformation function form the variables of this evaluation. For each data set, we obtain a large set of circle representations. We evaluate them individually by applying our potentially contradictory quality measures.

4.6.1. Methods

In the following, we present the investigated clustering methods, which assign points to clusters. In Section 3.4, we have introduced the different categories of clustering methods. For the evaluation we considered certain representatives of the four groups *Partitional Clustering*, *Distributional Clustering*, *Hierarchical Clustering* and *Density-Based Clustering*. Since we have already explained the principle of these methods in detail, we will only list the candidates here and provide details about their implementation.

Partitional Clustering

For *partitional clustering* we evaluate the k-Means algorithm since it is the most prominent algorithm for clustering. Furthermore, it provides satisfactory results in a variety of use cases. We use the k-Means++ implementation of Apache Commons Math[1]. The pluses indicate a modification of the initialization process. It uses a more advanced technique, which (empirically) improves the convergence rate, and thus the runtime in contrast to pure sampling. In contrast to k-Means, where the result depends on the initialization, it guarantees a solution that is $O(log(k))$ apart from the optimal k-Means solution [AV07]. The k-Means algorithm has one parameter, k, which specifies the number of output clusters. In our case this corresponds directly to the number of circles m after the transformation. To

[1] commons.apache.org/proper/commons-math/

compensate the deficit of not knowing the optimal k, we examined settings for k in the range between 1 and 15 000.

Distributional Clustering

For *distributional clustering* we evaluate Expectation Maximization (EM). We use the implementation of the well-known OpenCV[2] [KB13] library. From our experience, it provides the most stable implementation of the algorithm, which is robust against vanishing Gaussians that lead to singular variance matrices (making it difficult to continue the algorithm, i.e., many library implementations simply crash). EM has the same parameter k as k-Means such that k Gaussians are initialized upfront. Again, this parameter k ($k \in [1, 15\,000]$) corresponds to the number of circles m in our case when we apply the transformation function to the resulting clusters.

Hierarchical Clustering

For *hierarchical clustering* we incorporate multiple variants of agglomerative clustering. The reason for this is that each variant is superior to other ones for certain distributional patterns of data. The variants of choice are Single-Link (*SLINK*), Complete-Link (*CLINK*) and Centroid (*centroid*) clustering. The ELKI library[3] [Ach+12] provides many implementations of agglomerative clustering variants that refer directly to research papers. The two former variants are applied by *AGNES* [KR90] while the latter one uses *AnderbergHierarchicalClustering* [And73]. These three variants output a dendrogram which we utilize to extract clusters of cardinality m. We try values for m, similar to k-Means, in the range between 1 and 15 000.

Additionally, we implemented the Delaunay-based point aggregation algorithm *GeoTemCo* by Jänicke et al. [JHS13]. It essentially represents a hierarchical clustering algorithm. In contrast to the aforementioned methods, it directly outputs circles using circle sizes identically to those generated by the log_2 transformation. Thereby, this eliminates the process of creating a dendrogram and cutting out the desired number of clusters. The algorithm is directly applicable for visual point clustering.

[2] www.opencv.org
[3] elki-project.github.io

Density-Based Clustering

For *Density-Based Clustering* we evaluate the most prominent candidate, namely *DBSCAN* (Density-based Spatial Clustering of Applications with Noise) [Est+96]. We use the *DBSCAN* implementation of the ELKI library to generate clusters based on their mutual proximity. The algorithm utilizes the parameters $MinPts$ and ϵ which were already described in Chapter 2. Here it is not necessary to specify the number of output clusters. The number rather implicitly results from the density setting. While it seems reasonable to set ϵ to r_{min} and $MinPts$ to 1 (in order to not loose any points as outliers), we try values for $\epsilon \in [2.5, 25]$, and $MinPts \in [1, 10]$.

4.6.2. Data Sets

We selected five data sets of different sizes for the evaluation. Four address the domain of biodiversity research and concern species observations from GBIF (cf. Section 1.2). These are *Loxodonta cyclotis* (African forest elephant) with 23 points, *Puma concolor* (cougar) with 1 992 points, *Macropus giganteus* (eastern grey kangaroo) with 23 039 points and *Alnus Glutinosa* (common alder) with 185 259 points. We filtered the entire database with each species name in order to generate the data sets. The fifth data set includes point locations from OpenStreetMap[4]. They correspond to library locations in Germany (*german libraries*) that were extracted from points of interest. This data set contains 3 383 points and follows potentially different distribution patterns than the species observations.

4.6.3. Clustering Methods

For the first experiment, we only consider the first six quality measures and omit zoom consistency. We will consider this measure in a separate experiment. We calculated the results for all combinations of methods, parameter settings and quality measures. To present the results in a meaningful way, we consider only the Pareto frontier or skyline [Pap+05] of the results. This means that we discard all solutions that are dominated by another solution, i.e., that are better in each quality score. Using these frontiers, we can assure a fair comparison independent of the parametrization. For instance, if a Pareto frontier of one method is completely

[4] www.openstreetmap.org

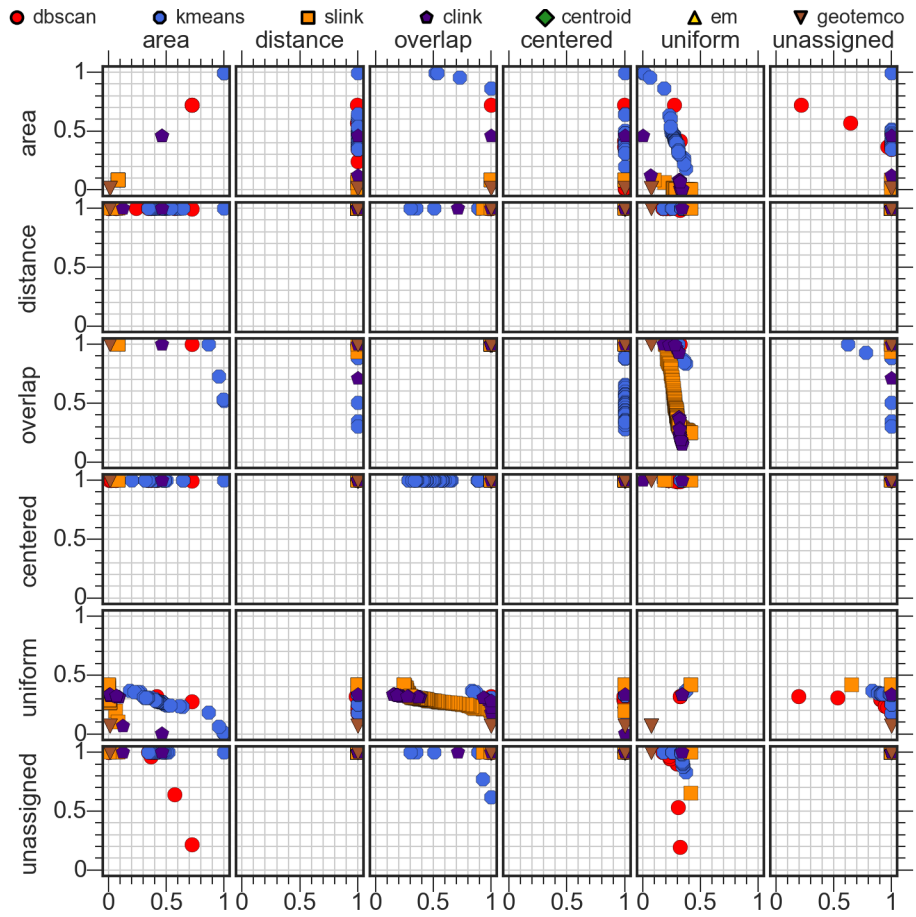


Figure 4.10.: This plot matrix shows all pairwise Pareto frontiers on data set Alnus glutinosa.

above the Pareto frontier of another method, it indicates that the first method outperforms the second one.

Figure 4.10 presents the results in a pairwise plot-matrix for the Alnus glutinosa data set. The other results are not shown here because they take up a lot of space (which inflicts readability), but are available in Appendix A. The x and y axes plot the quality measures against each other. The inner scatter plots (grid cells) show the quality value of the individual quality measures. The diagonal of the plot matrix depicts the individual criteria. The criteria *distance*, *overlap*, *centered* and *unassigned* can be fulfilled completely by all approaches. This means that there is at least one parametrization at or very close to the position $(1, 1)$, which indicating a perfect result.

The criterion *area* can only be completely fulfilled by k-Means. The other ap-

Table 4.1.: This table shows the dominance ranking for the projections for each data set. The numbers in braces show the amounts of dominated points.

τ	Alnus		Libraries		Loxodonta		Macropus		Puma	
cc	1	(574)	91	(306)	-		209	(2343)	137	(1660)
$\log_2$	139	(395)	5	(372)	-		1	(4182)	1	(2901)
$\log_{10}$	22	(528)	1	(512)	1	(1489)	29	(3176)	84	(1823)

proaches perform worse. One explanation for *DBSCAN* is that it suffers from non-convex clusters resulting from density-connected data points. A transformation into a circle most likely leads to errors in this case. Likewise, *SLINK* is much more susceptible to tubular clusters that are hard to approximate by a circle. On the other hand, *CLINK* has a better quality due to the different distance function which prefers circular clusters. *GeoTemCo* performs worse for this criterion. However, it is much better for all the other data sets that are significantly smaller. Obviously, the *log*$_2$-scaling poses a problem. Therefore, it is the subject of another experiment.

The *uniform* criterion is not sufficiently fulfilled by any method. This is because there are no uniformly distributed clusters in the underlying point data set. Therefore, it is difficult to fit uniform areas. *GeoTemCo* seems to have a slight deviation from the optimum because the circle is displaced in each step of the algorithm.

The plots that are not on the diagonal of the matrix show combinations of different criteria. The *uniform-overlap* plot shows an interesting curve that indicates that relaxing the overlap constraint makes it possible to better fit the circles to the uniform spots of the point data set. Such a phenomenon occurs when one chooses a large value for m, i.e., more circles are obtained that allow or facilitate this adjustment. If the user, for instance, focuses on the criteria *uniform* and *overlap*, there are different choices: While *SLINK* provides in at least one parametrization the best uniformity, *GeoTemCo* is the best choice for the criterion *overlap*. However, the user gets the best compromise by applying *SLINK* in different parametrizations. Similar tendencies exist for other combinations of criteria, but they are less pronounced.

4.6.4. Transformation Functions

The previous experiment showed that the choice of the transformation function might have a considerable impact on the quality score. In this experiment we

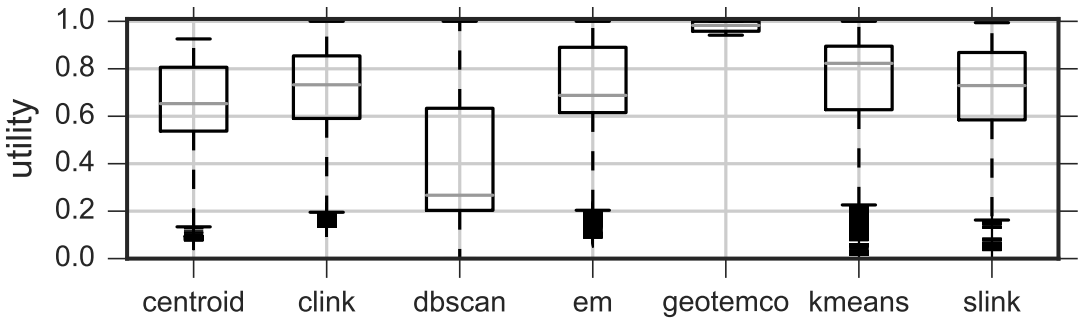


Figure 4.11.: This boxplot shows the different quality values regarding the zoom consistency.

evaluate the different transformation functions τ and summarize the results in Table 4.1. For each data set, each method and each τ function, we first calculate the number of dominated points within the Pareto frontier. The higher this amount, the better the transformation function works for this data set. Then, for each transformation, we obtain the best result according to the rank. For instance, in the case of the Alnus data set, *circumcircle* performs best for a certain method and parametrization. In more detail, it dominated 574 other approaches. The best method (and parametrization) using log_{10}, on the other hand, is ranked on position 22. This means that there are 21 other methods that use *circumcircle* that have performed better. Finally, log_2 performs worst.

For small data sets, transformations based on the logarithm clearly perform very well. However, if the size of the data set increases, it is necessary to reduce the maximum radii. This is possible, for example, by selecting larger bases for the logarithm, since the logarithm function itself does not automatically imply appropriate radii. If the radii are too large, problems (e.g. with uniformity) arise, which have been described in Section 4.6.3. In such scenarios the *circumcircle* approach performs surprisingly good, as in the case of the Alnus data set. This clearly indicates that the maximal circle areas have to be adjusted, for instance by incorporating the drawing area rather than just relying on the number of input points.

4.6.5. Zoom Consistency

Finally, we examine the seventh quality measure, the zoom consistency. For comparison, we chose three zoom levels to calculate the quality score. Three zoom levels are sufficient for our scenario to evaluate the zoom consistency. This is due to the transitivity of the successive levels.

Figure 4.11 shows the results in the form of boxplots. We can observe that all approaches are able to produce very good results since there is at least one parametrization for each approach that produces quality values that are close to 1. However, the majority of results is located inside the boxes, indicating that many parametrizations of *DBSCAN* lead to non-optimal zoom consistencies. Since parametrization is a crucial point for users, *DBSCAN* appears to be difficult to use. Other approaches are much more robust in this respect. *GeoTemCo* leads to nearly perfect results for our data sets. Since it does not require any parametrization, this is an important finding. It shows that this approach allows for appropriate use.

4.7. Multiclass Adaptations

Until now, the quality measures were limited to support the evaluation of circle representations of point data sets containing a single class. In this section, we extend the problem for the case where multiple classes exist. In particular, we introduce another indicator parameter cls, which specifies the class as an integer value. Consequently, this parameter is an additional attribute attached to a point and a circle (cf. Section 2.3). In the following, we examine the necessary changes for each measure of Section 4.4. In particular, we propose an alternative set of quality measures that is suitable for the problem of multiclass visual point clustering.

4.7.1. Assignments

Before presenting the multiclass quality measures definitions, we discuss how the two assignment functions are adapted.

Nearest Neighbor Assignment

We define the multiclass nearest neighbor assignment $\mathcal{N}^*$ of $c_i \in C$ as follows:

$$\mathcal{N}_i^* := \{p \in P \mid c_i = \arg\min_{c \in C} \ dist(c, p)\}$$

with

$$dist(c,p) := \begin{cases} \infty & \text{if } c.cls \neq p.cls \\ cpdist(c,p) & \text{if } cpdist(c,p) > 0 \\ -\sqrt{2} + pdist(c,p) & \text{else} \end{cases} .$$

By setting the distance to infinity when the classes of point and circle do not match, a point is only assigned to the nearest circle of its class.

Enclosing Assignment

The multiclass enclosing assignment $\mathcal{E}^*$ of $c_i \in C$ is defined as

$$\mathcal{E}_i^* \coloneqq \{p \in P \mid contains(c_i, p) \wedge c_i.cls = p.cls\} \ .$$

This ensures that circles can only enclose the points of their own class. The definition of *contains* is the same as before.

4.7.2. Measures

In the following, we revisit every quality measure once again and check how to adapt it to work in the multiclass problem setting. Since we have defined the multiclass nearest neighbor and the multiclass enclosing assignment to deal with multiple classes, the same deviation measures are applicable as before in the one-class case in most cases.

For *area proportionality* there is no need to change the density calculation and no need to change the calculation of the coefficient of variance. The same applies to *circle points centered* and *circle point distance* since the measure is averaged over all circles. This is still valid because of the modified assignment functions. Also for the *uniform point distribution* it is sufficient to calculate the uniformity for each circle and to calculate the mean value. The assignment ensures that the uniformity is calculated between points of the same class.

Unassigned points can make use of the multiclass enclosing assignment and its residual assignment. However, it is likely that the utility value will be lower compared to the one-class scenario. For example, if a good overlap rating is achieved, the circles of a single class cannot occupy the entire map space, but must segment the available space. This seems inevitable.

Circle overlap does not incorporate the points, but includes just the circles into the utility computation. The intersection and the union are calculated over all circles. However, this is still valid in the multiclass scenario. Since we want to penalize the loss of information in the form of occlusion, it is not necessary to consider the individual classes. Instead, we weigh each occlusion equally.

For *zoom consistency*, on the other hand, we have to adjust the measure. This is because zooming in should resolve circles into (potentially) smaller circles of the

same class. Here it is not useful to accept that a cluster splits into parts that refer to data that does not belong to that cluster. This means that we slightly extend the definition to

$$\nu^*_{zoom} := \frac{1}{m} \sum_{c \in C^{(z)}} \frac{1}{A(c)} \cdot A\left(\left(\bigcup_{\{c' \mid c' \in C^{(z+1)} \wedge c'.cls = c.cls\}} c' \right) \cap c \right) .$$

The adjustment basically only considers circles from $z+1$ that have the same class cls as the actually considered circle.

4.8. Summary

This chapter presented a set of quality measures for the evaluation of visual point clusterings of point data sets. We presented multiple visual assignment methods that perform the mapping from points to circles. We then defined the individual measures that build upon these assignments.

We showed experiments that incorporated existing clustering methods and evaluated them for their applicability for visual point clustering. Beforehand, we discussed different transformation techniques that map a set of points of a cluster to a circular representation. Within the experiments, we have seen that the parametrization of the different methods poses a problem for achieving good results. It is therefore difficult to set these parameters optimally in a particular application. With *GeoTemCo* we have found a suitable method that does not require parametrization. In addition, the choice of the transformation technique has a substantial impact on the visual point clustering quality for a given a data set.

For the multiclass scenario, we have proposed an adjustment of the criteria. Fortunately, due to the definition of the visual assignment functions, only minor adjustments were necessary to prepare the measures for the multiclass scenario.

5

CMQ: The Circle Merging Quadtree

This chapter presents the details of the Circle Merging Quadtree (*CMQ*), an algorithm that computes a visual point clustering from a set of input points. The chapter starts with an introductory motivation about the algorithm in Section 5.1. It discusses whether it is necessary to develop a new algorithm with respect to the findings of Chapter 4. Then Section 5.2 introduces the *CMQ* method by first describing the general idea for a better understanding and then giving a comprehensive explanation of the algorithmic details. It also contains pseudocode of all its important algorithms. Thereafter, Section 5.3 provides a discussion of the time and space complexity of *CMQ*. This gives insight into the strengths of the algorithm and theoretical bounds. Subsequently, Section 5.4 provides a discussion about performance improvements and the stability of *CMQ*. In particular, it provides an efficient preprocessing step of *CMQ* that guarantees stability and details on the effects on temporal and spatial complexity. Section 5.5 deals with the computation of subsequent zoom levels. It discusses possibilities to accelerate the computation with respect to the properties of *CMQ*. The experiments in Section 5.6 provide comprehensive evaluations of time, quality, stability and space aspects of *CMQ* and its competitors. It also describes the used data sets and gives an analysis that puts all results into perspective. Finally, Section 5.7 provides a brief summary of this chapter.

5.1. Motivation and Requirements

In Chapter 4, we have seen that it is complicated to find an algorithm for visual point clustering that provides a good quality. There are several aspects that an algorithm must cover in order to solve the problem adequately. General clustering methods have not proved to be well suited for the problem at hand. These methods are either laborious to parameterize (e.g. the number of clusters k in k-Means) or do not fit the approach of circular patterns (like *DBSCAN*). We were able to find good settings most of the time, but finding these settings is generally

difficult because they require a lot of computing time or a heuristic estimation. For instance, a systematic grid search requires computing numerous clusterings for different parameter combinations. On the other hand, a heuristic estimation of the parameters poses another source of error. *GeoTemCo*, as a specialized algorithm, provides good overall results with respect to our quality criteria. However, its runtime is $O(n \cdot log(n))$ and we will show in the experiments (Section 5.6) that it lacks in performance for a big data scenario.

Overall, it seems promising to develop a specialized algorithm that addresses the problem of visual point clustering and does not require non-trivial parametrization from the user. But it is also important to consider time constraints for the computation. Computational requirements involve responsiveness and throughput. The latter one is important to fit into a big data scenario. The former one is of utmost importance to enable interactivity. Interactivity is the key concept for facilitating exploratory scenarios. Here, as a recap of Section 3.2, users can browse through different data sets and examine interesting data, e.g. hot spots or outliers, in more detail. For the computational constraints, we have to look at both theoretical limits and the practical runtime performance of this algorithm.

Another interesting aspect is the rate of data compression that such a visual point clustering algorithm is able to achieve, while facilitating information recognition at the same time. Since an occluded view of the raw data on a map hides information, this is an advantageous side aspect. Data reduction is very important in mobile usage scenarios. This is the case because the mobile bandwidth is limited and the computing power of the consumer hardware is also limited, especially when considering the battery power. So it is precisely beneficial when less data needs to be transferred and displayed. Our experiments will investigate and measure this feature.

5.2. Method

In the following, we first describe the idea behind the Circle Merging Quadtree (*CMQ*). Then we elaborate on the fundamental algorithms in more detail. We discuss the important steps of inserting, querying and merging by providing pseudocode and give additional estimates regarding the size of the expected output.

5.2.1. Idea

The fundamental idea of *CMQ* is to work directly with circles rather than points. Although the input is a set of points in the visual point clustering scenario, the output is a set of circles. On a screen, we typically display points (which have no area) as circles with a radius (e.g. r_{min} in our case). Therefore, it makes sense to consider the problem of visualizing a set of circles instead. This means that we have to transform all input data into circles beforehand. The subsequent section will provide more details here.

CMQ treats the overlapping of circles as the main issue. Therefore, it will enforce an overlap-free output while simultaneously preserving other spatial properties as much as possible (e.g. area proportionality and the centering of circles with respect to their represented points as reflected in the quality measures in Chapter 4). Moreover, unlike other algorithms, *CMQ* does not displace circles in order to reduce overlap that could lead to an additional bias in the result. In addition, *CMQ* does not omit any data. Thus, it deliberately preserves outliers and does not concentrate only on the aggregation of hot spots.

For reasons of efficiency, *CMQ* employs a spatial index structure that comes with the following three properties. First, it allows storing circles rather than points or rectangles. Second, it efficiently supports circle queries to detect overlaps. Third, the index structure is dynamic, i.e., it supports insertions and deletions of circles.

The core strategy of *CMQ* is to find overlaps as quickly as possible and to immediately merge circles. This leads to an index structure that only needs to store a small number of elements. In addition, *CMQ* guarantees that the circles that are kept in the index are non-overlapping. Rather than inserting all the base circles of the points in advance, *CMQ* resolves the overlap at the insertion time of each new circle. Fewer objects in the index structure have two advantages. First, this results in faster insertion and query times. Second, fewer objects allow processing large data sets with less main memory, and thus, it leads to more efficient memory usage in general. In Section 5.4.2, we detail on making the overlap checks even more efficient by introducing a grid-based preprocessing step.

With respect to the quality measures in Chapter 4, the size of the circles needs to reflect the number of attached points. In Section 4.5, we discussed techniques that allow a proportional scaling of the circles. In fact, the size of the circles grows monotonically in the number of attached points, so that merges of circles always lead to larger circles. Moreover, the radii of the circles are restricted to a range between r_{min} and r_{max}. The lower bound is useful for the following reasons. First,

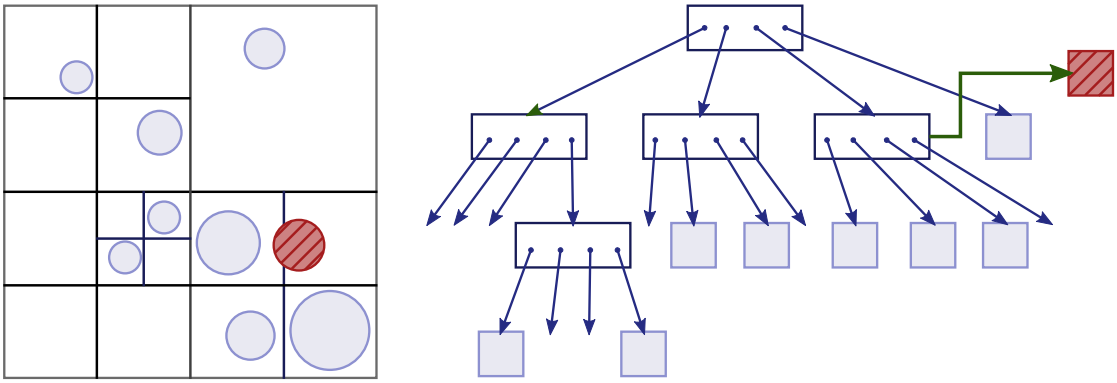


Figure 5.1.: This figure shows an exemplary quadtree partitioning of a set of circles (a) and the corresponding tree structure with pointers to internal and leaf nodes (b).

it allows interactive users to select circles easily via a mouse click. Second, it is important to derive an upper bound for the number of circles that are kept in the index of *CMQ*.

CMQ uses the (natural) input order of the data as input. Therefore, it does not require any sorting as a preprocessing step. This is very advantageous for the worst-case runtime (see Section 5.3). However, without sorting the input, there is no stable visualization of the circles, which will be discussed in Section 5.4.3.

With respect to the classification of clustering algorithms, *CMQ* is a hybrid of hierarchical and density-based clustering. Like hierarchical clustering, it first merges circles with small distances, but does not give any guarantee that this occurs in the order of the distances. Like density-based clustering, it inevitably merges overlapping circles, i.e., circles in areas of a high density. However, it represents each of the clusters by a circle and not by a complex polygon that covers all the points of the cluster.

5.2.2. Algorithm

The goal of this subsection is to present the index structure and its important algorithms. The index structure is an adaption of the point-region quadtree (cf. Section 3.3). As for the standard quadtree, each internal node partitions the corresponding space into four quadrants. The space of the root node corresponds to the bounds given by the map. We extended this basic quadtree to be able

Algorithm 5.1 INSERT

```
    Input: A quadTree tree data structure
           A circle to insert
 1: currentNode ← ROOT(quadTree)
 2: while HASCHILDREN(currentNode) do
 3:     for child ∈ CHILDREN(currentNode) do
 4:         if ENCLOSES(child, circle) then
 5:             currentNode ← child
 6: while ISLEAF(currentNode) ∧ ISFULL(currentNode) do
 7:     children ← SPLIT(currentNode)
 8:     // pass circles of currentNode to children
 9:     for child ∈ children do
10:         if ENCLOSES(child, circle) then
11:             currentNode ← child
12: ADD(currentNode, circle)
```

to cope with circles instead of points. Recall the definition of a circle as a four-tuple (x, y, r, n), where the parameter n represents the number of points that are attached to the circle. In our version, a circle is stored in the lowest tree node that fully contains it.

The quadtree consists of leaf nodes of constant size and index nodes with a fan-out of four. In the following, we assume that at most one circle fits into a leaf node. If a leaf node consist of more than one circle, the quadtree recursively subdivides the associated cell into four equal-sized child cells until all cells at maximum contain one circle. In contrast to points, this is not always possible because a circle can intersect with a cell boundary. In our adapted version, we store such a circle in a list associated to the lowest internal node whose cell still encloses the circle. This is illustrated in Figure 5.1, where circles are stored in a quadtree. On the left hand side (a), the partitioning of the quadtree is plotted, whereas the tree is shown on the right hand side (b). Here, all circles except the red-marked one fit into one leaf. The red-marked circle is stored in a list associated with the lowest internal node that fully contains this circle. This organization introduces no copies of circles, and thus, the operations for insertions, deletions and exact match search are restricted to a single path only. In addition, the maintenance of copies would have increased the storage overhead. Before we introduce the complete *CMQ* algorithm, we first describe the functions INSERT and QUERYANDEXTRACT of the adapted quadtree.

INSERT Algorithm 5.1 shows the pseudocode of the insert function of the adjusted circle quadtree. It requires a quadtree as input as well as a circle that should

Algorithm 5.2 queryAndExtract*

Input: A *quadTree* tree data structure
A *query* circle to test for overlaps
Minimum inter-circle distance δ in px
Output: A set of overlapping circles

```
1: nodes ← createStack( )
2: push(nodes, head(quadTree))
3: while ¬isEmpty(nodes) do
4:     currentNode ← pop(nodes)
5:     for circle ∈ circles(currentNode) do
6:         if intersects(query, circle, δ) then
7:             remove(currentNode, circle)
8:             return {circle}
9:     for child ∈ children(currentNode) do
10:        if intersects(query, bounds(child), δ) then
11:            push(nodes, child)
12: return ∅
```

be inserted into the tree. The first part (lines 1-5) tries to find a node of the tree into which the new circle fits. To do this, we traverse the tree in a recursive fashion by starting from the root (line 1). For each node, the four child nodes are examined whether one of them encloses the circle (line 3). If so, the insertion proceeds in the corresponding subtree. Otherwise, the insertion stops either at a leaf node or at an internal node that is the deepest one that still encloses the circle.

The second part of the insert algorithm (lines 6-11) considers that all leaf nodes of the quadtree have a capacity threshold on which they split. Hence, as long as the *currentNode* is a leaf node and *full* (line 6), it is divided into four quadrants and the node data (circles) is descended to the child nodes (line 8). Then, the insertion position for the circle is one of the child nodes (as long as it does not get stuck at this level, as discussed before). Since all circles of level l could theoretically descend into a single child node of level $l + 1$, it is necessary to check this in a loop until all conflicts are resolved. Finally, in line 12 the algorithm stores the circle at the previously determined node.

QUERY Algorithm 5.2 as well as Algorithm 5.3 describe the queryAndExtract method of the circle quadtree. They receive the quadtree, a query circle and the minimum inter-circle distance δ as input and outputs a set of overlap-

Algorithm 5.3 queryAndExtract

Input: A *quadTree* tree data structure
A *query* circle to test for overlaps
Minimum inter-circle distance δ in px
Output: A set of overlapping circles

```
1:  result ← createList( )
2:  nodes ← createStack( )
3:  push(nodes, head(quadTree))
4:  while ¬isEmpty(nodes) do
5:      currentNode ← pop(nodes)
6:      for circle ∈ circles(currentNode) do
7:          if intersects(query, circle, δ) then
8:              remove(currentNode, circle)
9:              add(result, circle)
10:     for child ∈ children(currentNode) do
11:         if intersects(query, bounds(child), δ) then
12:             push(nodes, child)
13: return result
```

ping circles from the tree. Note that we have introduced two variants of the algorithm only to facilitate determining the runtime of *CMQ* in Section 5.3. While queryAndExtract (without star) returns all overlapping circles, queryAndExtract* solely returns a single overlapping circle, since it immediately returns in case of a match (see line 8). Without specifying a specific version, we refer to the second version of queryAndExtract that returns all overlapping circles.

The queryAndExtract function starts with the initialization of a result list (line 1) and a stack for the node traversal (line 2). For the depth-first traversal, we first move the root of the tree to the stack (line 2). As long as there are nodes on the stack, we expand the tree and check for overlapping circles (lines 4-12). At each iteration, we retrieve a node from the stack (line 5). First, we check for all circles (line 6) that are stored in this node whether they overlap with the query circle with respect to the inter-circle distance (line 7). In the case of an overlap, the element is immediately removed from the tree and added to the result set (lines 8-9). This means that within the query function the removal operation poses a side effect. We do not include underflows in the algorithm where previously split nodes would be merged together. In the second step, the function expands the current node (line 10). The function places each overlapping child node on the stack for further

Algorithm 5.4 CMQ

Input: Set P of n points with coordinates x and y
Minimum circle radius r_{min} in px
Minimum inter-circle distance δ in px
Map bounds $B \in \mathbb{R}^2 \times \mathbb{R}^2$ in px
Maximum circle radius r_{max} in px
Output: A set of non-overlapping circles

```
1:  quadTree ← initialize empty tree with bounds B
2:  for point ∈ P do
3:      circle ← MAKECIRCLE(point.x, point.y, r_min, 1)
4:      overlappingCircles ← QUERYANDEXTRACT(quadTree, circle, δ)
5:      while overlappingCircles ≠ ∅ do
6:          circle ← MERGE(overlappingCircles ∪ {circle}
7:                         r_min, r_max, n)
8:          overlappingCircles ← QUERYANDEXTRACT(quadTree, circle, δ)
9:      INSERT(quadTree, circle)
10: return GETALLCIRCLES(quadTree)
```

expansion (lines 11-12). Finally, the function outputs the resulting set of overlapping circles.

Algorithm 5.4 provides the details of the main function CMQ. The input consists of the set of points P and the minimum circle radius r_{min}. Furthermore, it contains the minimum inter-circle distance δ, the map bounds B and finally the maximum radius r_{max}. Note that the application has to specify r_{max} using a heuristic like $r_{max} = 4 \cdot log_2(n+1)$, which was introduced as the Log_2 model in Section 4.5. We will use these bounds r_{min} and r_{max} in the merge step to scale the area of the circles c_i proportionally to their relative number of attached points $c_i.n$.

First of all, the algorithm initializes an empty quadtree with the bounds B of the considered map (line 1). Then it traverses over all points $p \in P$ of the input one after the other (line 2). The algorithm creates for each point p of the input a circle (four-tuple) with center $(p.x, p.y)$ and radius r_{min} by using the MAKECIRCLE function (line 3). This initial circle represents a cluster with exactly one point, and thus, the fourth parameter (cardinality) is set to 1. Thereafter, the algorithm tries to insert this circle into the quadtree while repeatedly checking for overlaps. Lines 5 to 8 provide the iterative lookup for detecting overlapping circles and merging them. We perform the lookup for overlaps as a radius query to the adjusted quadtree as described in Algorithm 5.3. The QUERYANDEXTRACT function does

Algorithm 5.5 Function MERGE for merging circles

Input: Set of circles $\{c_1, \ldots, c_k\}$ with attributes x, y, r and n
Minimum circle radius r_{min} in px
Maximum circle radius r_{max} in px
The total number of points n

Output: A new, merged circle c with attributes x, y, r and n

1: newSize $\leftarrow \sum_{i=1}^{k} c_i.n$

2: newX $\leftarrow \dfrac{\sum_{i=1}^{k} c_i.x \cdot c_i.n}{\text{newSize}}$

3: newY $\leftarrow \dfrac{\sum_{i=1}^{k} c_i.y \cdot c_i.n}{\text{newSize}}$

4: fraction $\leftarrow \dfrac{\text{newSize} - 1}{n - 1}$

5: $A_{min} \leftarrow {r_{min}}^2 \cdot \pi$

6: $A_{max} \leftarrow {r_{max}}^2 \cdot \pi$

7: $A_{new} \leftarrow A_{min} + \text{fraction} \cdot (A_{max} - A_{min})$

8: newRadius $\leftarrow \sqrt{\dfrac{A_{new}}{\pi}}$

9: **return** MAKECIRCLE(newX, newY, newRadius, newSize)

not only include the circle radius, but also a padding parameter specified by the inter-circle distance δ. If the query returns a result, the matching circles are merged with the current circle (line 6). Recall that the QUERYANDEXTRACT function directly removes all overlapping circles from the tree as a side effect. We repeat this procedure with the new merged circle until there is no more overlap. Then, the algorithm inserts the circle into the quadtree (line 9).

We can increase the efficiency of the insertions by introducing a technical modification to the function QUERYANDEXTRACT. Instead of returning an empty set in the case of no overlap, the algorithm returns insertion node that fully contains the query circle. We determine the insertion node by traversing the query path from the leaf, which contains the centroid of the query, upwards until that circle is completely enclosed by the bounds of this node. By doing this, we can improve the efficiency of the INSERT algorithm because most of the inserts occur in leave nodes and it can omit a second tree traversal.

The merging of multiple circles is realized in the MERGE function. It is given as pseudocode in Algorithm 5.5 and also illustrated in Figure 5.2. The input consists

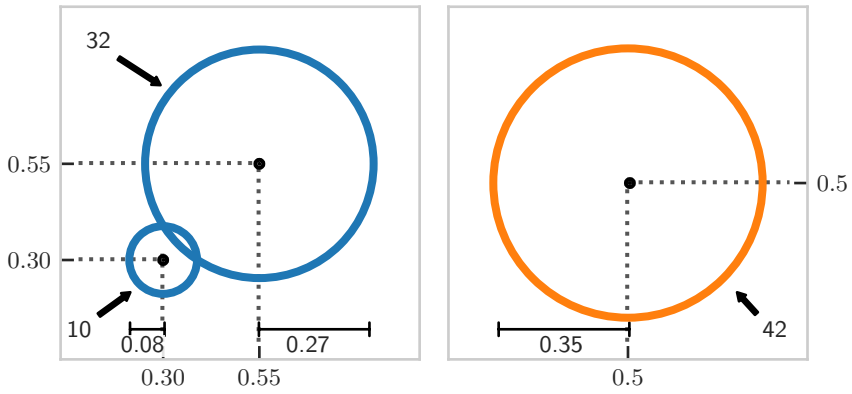


Figure 5.2.: This figure illustrates the merging of two circles (left) into a new, larger one (right). The centers and radii of the circles are given, indicating the growth and displacement after merging. The numbers at the arrows give the amount of attached points for each circle. The required calculations are given in Algorithm 5.5.

of a set of k circles to be merged as well as the parameters r_{min}, r_{max} and the total number of points n. The function returns a new merged circle. The lines 1 to 3 compute the center of the new circle as the weighted centroid of the input circles, where the weights are the relative number of corresponding points. In an analogous manner, the new area, which is determined in lines 4 to 7, is scaled linearly between the minimum and maximum area. This procedure is similar to the method described by Jänicke et al. [JHS13] except that we compute it for more than two circles. Eventually, in line 8, we derive the new radius from the area. In line 9, the function outputs the new circle.

After the *CMQ* algorithm has processed each of the points, the quadtree consists of a set of m non-overlapping circles. In line 10 of Algorithm 5.4, the function getAllCircles retrieves all circles from the quadtree and outputs them as the result. Here, a simple tree traversal is sufficient. The size of this set is bounded by the size of the input set ($|P| = n$) and the maximum number of displayable circles on the map. For a given zoom level z, an upper bound for the number of displayable circles is given by

$$\frac{map\_area}{min\_circle\_area} = \frac{256^2 \cdot 2^{2z}}{r_{min}^2 \cdot \pi} .$$

This bound does not yet incorporate δ and unavoidable gaps between circles. Furthermore, we expect m to be much smaller than this upper bound due to a high number of expected overlaps.

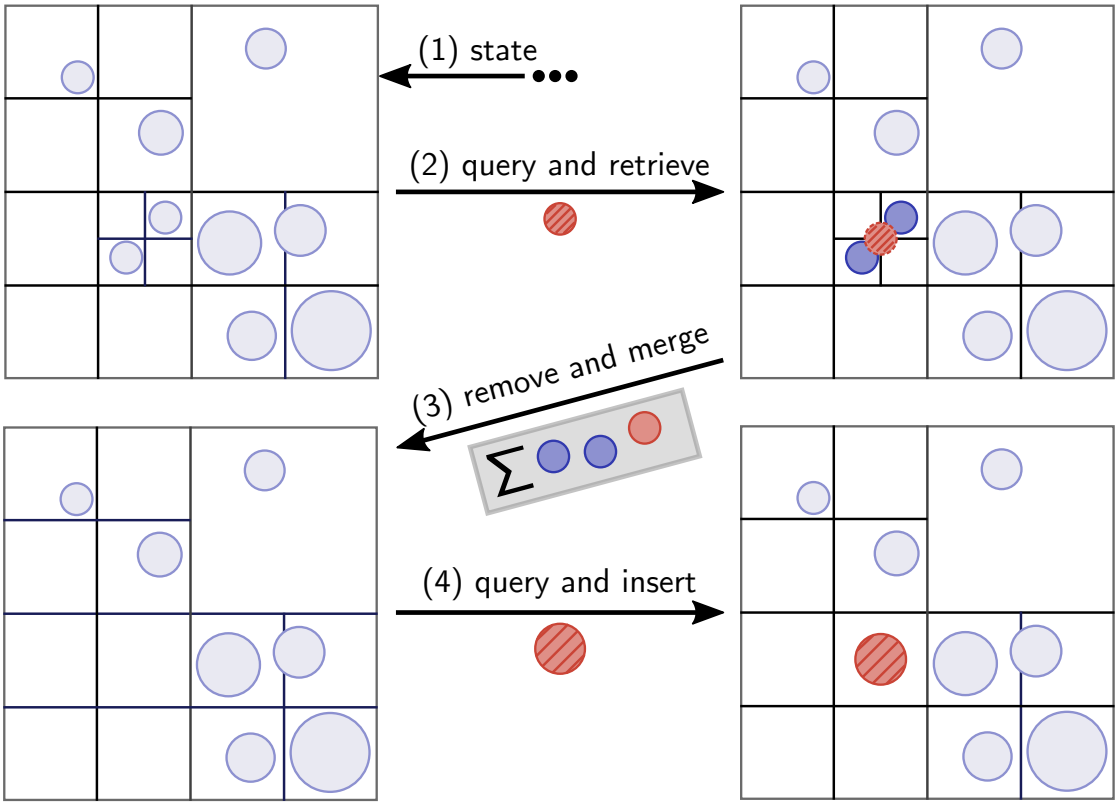


Figure 5.3.: This figure shows an example of a *CMQ* insertion of a point. After querying with the minimum circle at the point's center (red and marked), the overlapping circles are removed from the quadtree (dark blue). These circles are then merged into a new, larger circle that is then inserted into the quadtree.

Figure 5.3 presents an example of the insertion phase of *CMQ*, starting (1) with the quadtree from Figure 5.1. Here, we restrict the example to only plotting the partitioning of the quadtree. The red-marked circle, which is a converted point with minimum radius r_{min}, serves as a query to the quadtree. The query (2) outputs two overlapping circles (dark blue), which are merged with the query circle. Note that during the merge step, the tree no longer contains the previously overlapping circles (3), and thus, a reorganization of the tree is performed. Then, the newly created circle serves as input to the next query to the quadtree. Since there is no overlap anymore, the algorithm inserts the circle into the quadtree (4). This completes the insertion step of the point and the algorithm can now continue with the next point. In the next section, we examine the time and space complexity of this algorithm.

5.3. Time and Space Complexity

In this section, we investigate the theoretical complexity of the *CMQ* algorithm. For this, we first consider the complexity of the runtime and secondly look at the space and memory consumption.

5.3.1. Time Complexity

To analyze the time complexity, we first look at the individual operations of the *CMQ* quadtree. Then, we analyze the runtime of each quadtree operation. Finally, we can combine the results to present the total runtime of *CMQ*.

Line 1 of Algorithm 5.4 takes constant time for the empty tree structure being initialized with its boundaries B. The for-loop in line 2 ensures that there are exactly n calls of makeCircle (line 3), queryAndExtract (line 4) and insert (line 9). While makeCircle requires constant time since it only initializes a four-tuple, we will investigate queryAndExtract and insert later. The remaining operations to estimate are in lines 6 to 8, since the condition of the while-loop in line 5 causes only constant costs. By using the queryAndExtract* algorithm (cf. Algorithm 5.2), we guarantee that it outputs either no or exactly one circle that overlaps with the query circle. The overlapping circle is then merged in the next iteration by using the operation merge in line 6. However, there is an upper bound of $n-1$ merge operations. There are two extreme cases. First, there are no overlaps at all and the loop is completely ignored. Second, there is only one circle left after $n-1$ merge operations. Therefore, the total upper bound of the operations insert, queryAndExtract* and merge is

$$1 + \sum_{i=2}^{n} 2 = 2n - 1 = O(n) \ .$$

The same bound holds when using the queryAndExtract variant without star (cf. Algorithm 5.3). Here, the number of loop iterations is less than or equal to the starred variant because it can output more than one circle to merge. This leads to the same merge operations at most.

Before we examine the runtime of insert and queryAndExtract, we show that the height of the quadtree is $O(1)$. The height of the tree is not bound by the number of elements n, but rather by the map width $w = 256 \cdot 2^z$. Without loss of generality, we consider a quadratic map of size w^2. The node width decreases exponentially with the depth of the tree because the width is halved for each level. Therefore, a circle with a minimum radius of r_{min} does inevitably exceed the area

of the node boundaries at a certain level in the quadtree. This leads to a height bound of

$$O\left(\frac{log\,(w)}{log\,(2 \cdot r_{min})}\right) .$$

This bound is much smaller than the worst-case depth of a quadtree, which is multiplicatively composed of a linear component in n and the logarithmic fraction between the root node length and the smallest distance of two points (cf. Section 3.3). Since we incorporate the map bounds as a constant for a call of the *CMQ* algorithm, the height is $O(1)$.

Subsequently, the time complexity of a single INSERT operation (cf. Algorithm 5.1) is $O(1)$. Since the function traverses a path from the tree root to a leaf node (lines 2-5) and the height is $O(1)$, it requires a constant number of operations. Furthermore, the time required to split an overfull leaf node (lines 7-8) is $O(1)$ because the algorithm has to split up a constant number of circles into a constant number of child nodes. Circles that cannot descend into the tree because they overlap the borders of a node are stored in the associated list of a tree node. However, the list size is also bounded by the width of the map due to the extent of the circles. There can only be a fixed number of circles assigned to a node before they inevitably trigger a merge operation. Aside from that, the operations ROOT, HASCHILDREN and ENCLOSES require constant time.

The QUERYANDEXTRACT operation can be analyzed similarly to the INSERT operation. Here, the constant depth of the tree leads to a constant number of nodes that are traversed in the worst case. Hence, the operation requires $O(1)$ time for both algorithms (cf. Algorithms 5.2 and 5.3). One can argue that in QUERYANDEXTRACT the `result` list potentially consists of all n circles. However, all QUERYANDEXTRACT operations in total only visit at most n circles since the merge operations would then reduce them to a single circle. Therefore, for the variant QUERYANDEXTRACT, the operations require $O(1)$ time on average. Moreover, the operations CREATELIST, CREATESTACK, ISEMPTY, POP, PUSH, CIRCLES, INTERSECTS and CHILDREN require constant time. They are either trivial or only perform operations at the front or back of a list or stack. The function REMOVE has to delete circles out of a list with constant length on average.

Finally, we examine the cost of the merge operations (cf. Algorithm 5.5). While the calculations in lines 1 to 3 require $O(k)$ time for merging k circles, there are at most n of such operations overall. The reason is, as discussed in the analysis of the QUERYANDEXTRACT operation, that after $n - 1$ merges of circles there is only one circle left. All other computations from lines 4 to 9 as well as the MAKECIRLCE method require constant time. Therefore, MERGE requires on average $O(1)$ time.

A side observation is that in the worst case, when all circles overlap, the costs for the query and insert operations are minimal since there is only one circle in the tree at a time. Vice versa, having no overlaps and a maximum number of circles in the tree yields a minimal number of query and insert operations.

5.3.2. Space Complexity

The analysis of the space complexity is, in contrast to time complexity, rather straightforward. Since the only data structure is the quadtree that stores the circles, we again examine the two extreme cases in the *CMQ* algorithm (cf. Algorithm 5.4). In the first case, if no circles overlap at all, there are at most n circles stored in the tree at the end. However, as discussed in Section 5.2.2, overlaps are inevitable for a large n because it is only possible to display less than $\frac{w^2}{r_{min}^2 \cdot \pi}$ circles side by side on a map of width w. Note again that this bound does not incorporate the inter-circle distance δ. Hence, together with potential pointers for the tree structures (loosely estimated with one pointer per circle) the space complexity is $O(w^2) = O(1)$, assuming w being a constant. The second extreme case considers the option that all circles overlap. Therefore, there is at any time exactly one circle stored in the quadtree and the bound of $O(1)$ also applies here. Since queryAndExtract removes all circles from the quadtree that it returns, it does not affect the space consumption. In any case, there are not more than $O(1)$ circles in the result set of queryAndExtract.

In summary, the combination of $O(n)$ time and $O(1)$ space complexity makes *CMQ* a suitable candidate for a big data scenario. The experiments in Section 5.6 confirm this for the actual implementation with real data. Since the height of the quadtree constitutes a constant in the runtime complexity, it is worth reducing this constant in order to improve the practical efficiency of *CMQ*. In the next section, we will show that we can further increase the efficiency by introducing a preprocessing step.

5.4. Preprocessing and Stability

In this section, we present a preprocessing step for the Circle Merging Quadtree that serves two purposes. It improves the performance of our method and produces stable results, i.e., the output is independent of the ordering of the points. In the following, we first motivate the importance of stability. Then, we present

the details of our preprocessing method. Finally, we analyze its time and space complexity.

5.4.1. Stability

CMQ is a visual point clustering algorithm that aggregates point data. It solves the unsupervised learning problem of creating circular representatives of the underlying point data. There are two characteristics that are important for this type of analytical algorithm: determinism and stability. A deterministic algorithm produces the same result every time it is applied to the same input. This creates reliability for the user, as the results can be reproduced at any point in time. Although not every analytical algorithm is deterministic, it is very advantageous to achieve this property. For example, spring layouts [Di +94] are applied to display and arrange graph structures using different random seeds so that users can refresh the visualization until it produces sufficient results. Stability means that the algorithm, regardless of the order of input data, always produces the same result. This property is even stricter than determinism. Again, it is quite common for data analysis tools not to provide stability, e.g. BIRCH Clustering [ZRL96], where the tree structure depends on the order of the data. However, both properties allow the user to easily reproduce the results of analyses and are therefore a requirement in many applications. In particular, for visualization it seems to be important to provide stable results. Keim et al. [Kei+08b] pointed out that the acceptance of the presented results is essential. Also from our experience in GFBio, users get confused when results change because of a different ordering of the data.

It is straightforward to conclude that *CMQ* is a deterministic algorithm. By reconsidering Algorithms 5.1 and 5.3 to 5.5 of Section 5.2 we can state that there is no randomness in any of the proposed functions. The reprocessing of the same input data inevitably leads to the same iterations, and thus to the same results.

On the other hand, *CMQ* is not stable by default. In Algorithm 5.4, the order of the merge operations depends on the order of the input data. In each merge step, the circle center moves to the combined centroid of the input circles and the size scales with respect to a minimum and maximum radius (cf. Algorithm 5.5). This means that, considering three circles A, B and C, it can potentially make a difference which two circles are merged first. Figure 5.4 provides such an example where the merge sequence on the left leads to a different result than the merge sequence on the right side. On the left side, the merging of the circles A and B leads to a circle (green) that does not overlap with C. In contrast, merging

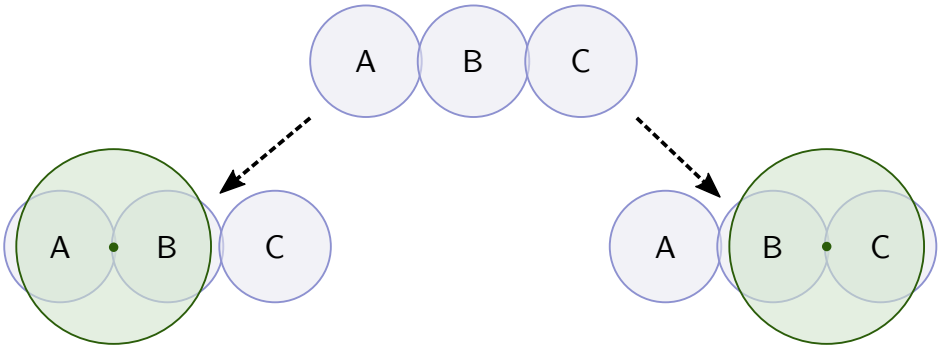


Figure 5.4.: This figure shows an example that a different order of circle merges can lead to a different result. The left side depicts that the merge of A and B leads to a circle (green) that does not overlap with C. The right side illustrates that the merge of B and C leads to a circle (green) that does not overlap with A. Thus, the left merge order leads to two different circles than the right one.

the circles B and C yields a circle (green) that does not overlap with A on the right side. As a result, there will be two different circles as result depending on the merge order. However, the fact that an algorithm is not stable is not always reflected in drastically different results. We will investigate the severity of this in our experiments in Section 5.6.

5.4.2. Preprocessing

In the following, we propose a preprocessing step that improves the efficiency of *CMQ* and also guarantees its stability. The core idea is that this step sorts the data prior to the insertions of the data into the quadtree. Thereby, it already identifies a large portion of the overlapping circle. These overlapping circles are immediately merged and only the remaining circles must be inserted into the quadtree. Furthermore, since we have evaluated the determinism of the *CMQ* algorithm and its quadtree modifications in the previous subsection, this sorting is sufficient to ensure stability.

Figure 5.5 illustrates the definition of a grid structure that has cells of width

$$\sqrt{2} \cdot r_{min} + \frac{1}{\sqrt{2}} \cdot \delta \;\times\; \sqrt{2} \cdot r_{min} + \frac{1}{\sqrt{2}} \cdot \delta \; .$$

In the algorithm, we will assign each circle to the grid cell that contains its centroid. This ensures that whenever at least two circles fall into the same grid cell, they do

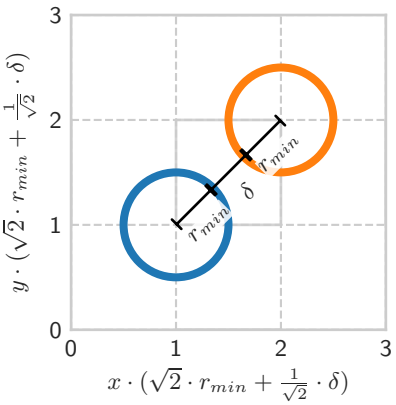


Figure 5.5.: The figure depicts the grid partitioning of the map for the preprocessing method.

overlap. Note that this merging would also occur in the *CMQ* algorithm without preprocessing if these circles were contiguous in a particular input sequence. The cell specification takes into account that overlapping means that the distance is at most δ between the circles. In Figure 5.5, the two circles (blue and orange) are placed at the furthermost distance within a single cell to depict this. In this case, the two circles are considered overlapping because they are just within the distance δ.

We describe the extended *CMQ* algorithm with preprocessing in Algorithm 5.6. The preprocessing starts with the creation of the grid of size k (line 1) where each grid cell $c \in \{c_1, \ldots, c_k\}$ stores a corresponding circle (x, y, r, n), which is initially undefined. The preprocessing then iterates over all data points (line 3), creates a circle with a minimum radius of r_{min} (line 4) and assigns it to the matching grid cell c_i (line 5). It updates c_i in line 7 by calling the MERGE function (cf. Algorithm 5.5) on the corresponding and new circle in order to update the combined centroid, radius and cardinality. The overall result of the merges is independent of the insertion order because all merge operations lead to a combined centroid that cannot exceed the grid cell. In the case that c_i's circle is uninitialized (line 9), it is directly populated with the circle of this iteration. In the penultimate step in line 10, the preprocessing retrieves all circles from the grid cells by traversing the grid. Finally, in line 11, we call *CMQ* with the result of the preprocessing. Note that we use the function CMQ˜, which is a modified version of *CMQ*. Here, the input is a set of circles instead of a set of points. It is then sufficient to skip the calls to MAKECIRCLE in *CMQ*.

There are two aspects that are important to ensure the stability of *CMQ* with

Algorithm 5.6 Function CMQ with preprocessing

Input: Set P of n points with coordinates x and y
Minimum circle radius r_{min} in px
Minimum inter-circle distance δ in px
Map bounds $B \in \mathbb{R}^2 \times \mathbb{R}^2$ in px
Maximum circle radius r_{max} in px
Output: A set of non-overlapping circles

```
1: grid ← initialize empty grid with bounds B and k cells of size
2:         √2 · r_min + (1/√2) · δ  ×  √2 · r_min + (1/√2) · δ
3: for point ∈ P do
4:     circle ← makeCircle(point.x, point.y, r_min, 1)
5:     c_i ← determineCell(grid, circle.x, circle.y)
6:     if isInitialized(c_i) then
7:         c_i.circle ← merge({ c_i.circle, circle }, r_min, r_max, n)
8:     else
9:         c_i.circle ← circle
10: circles ← gridCollect(grid)
11: return CMQ˜(circles, r_min, δ, B, r_max)
```

preprocessing. First, the grid structure does not merge overlapping circles that have centroids in different grid cells. There will be circles in several grid cells that overlap because their radius exceeds the cell boundaries, but they will be merged later in *CMQ*. Second, if we iterate over all grid cells in a stable order (line 7), the result of *CMQ* is also stable. This applies because the result is used as input for *CMQ* and *CMQ* is deterministic. Therefore, we must iterate over all grid cells in a predefined order and obtain the set of circles that corresponds to those cells. A space-filling curve like the *Hilbert*, *Snake*, *XY* and *Z* curve (cf. Section 3.3) provides such a stable order. It linearizes our two-dimensional map into one-dimensional indices. Since our grid provides a discrete partitioning of the map, we fulfill the requirements for the application of such curves.

In the following, we discuss the choice of data structure for the grid. There are basically two options for storing the grid structure: as an array or as a hash map. The former allows for fast insertions and provides stable iterations by default. It could pose, however, a potential overhead if the grid is very sparse because of skewed or concentrated data in the map extent. In this case, a grid uses much more memory than necessary, which can result in many accesses of empty grid cells, and thus lead to slower processing times. A hash map addresses this problem by being space efficient while offering constant expected access times at the same time. In this case, the space-filling curve serves as the prehash function. However, a good

hash function, which distributes the values uniformly, leads to few collisions, but also to a lack of locality. In order to iterate in a stable way over the hash table, it is necessary to use separate chaining for resolving collisions [Cor+09]. Open addressing techniques like linear probing or double hashing have the problem that the colliding entry that comes second in the insertion order is stored at a different position in the hash table. Traversing the underlying hash table would therefore result in circle lists that again depend on the insertion order. In contrast, for separate chaining, it is sufficient to sort the entries of the buckets according to the value of the space-filling curve to obtain stability during the iteration over the hash table. Since the number of entries in a bucket is close to 1 on average in a good setup [Kon10], sorting does not pose a significant overhead. Note that it is impossible to have two entries with the same key (space-filling curve value of the grid cell) because these entries would have been merged during grid insertion or update. This means that we do not have to consider the stability of the sorting algorithm. The experiments in Section 5.6 verify the findings and show the performance improvements to the unstable variant of *CMQ*. In addition, it contains measurements regarding the type of data structure and the type of space-filling curve in order to assess differences.

5.4.3. Time and Space Complexity

In this subsection, we will briefly discuss the time and space complexity of the preprocessing step of *CMQ*. In general, sorting algorithms require $O(n \cdot log(n))$ time. However, our grid data structure with the insertions and the retrieval is equivalent to a bucket sorting algorithm [Knu98]. This algorithm is able to sort the data in $O(n)$ time because the number of buckets is considered a constant. In this case, the grid divides the (constant) map into cells that depend on the minimum circle size (determined by r_{min}) and the inter-cluster distance δ. Therefore, we consider the number of buckets to be a constant. The choice of data structure (array or hash map) ensures that the access time is either constant or constant on average.

For space complexity, we need to look at the size that equals the maximum amount of data stored in the grid. When using an array, this size (= *number of cells*) again depends on the size of the map and the minimum circle. For the hash map, the number of stored grid cells is at most as large as for the array variant. In practice, it should be smaller if the data is not uniformly distributed. In both cases it is bounded by the map size rather than the number of elements n. Furthermore, we consider this to be much smaller than n in a big data scenario.

Algorithm 5.7 Multiple Zoom Level Aggregation

Input: Set P of n points with coordinates x and y
Minimum circle radius r_{min} in px
Minimum inter-circle distance δ in px
Map bounds $B_{crs} \in \mathbb{R}^2 \times \mathbb{R}^2$ in coordinate system units
Maximum circle radius r_{max} in px
Zoom levels $z_1, \ldots, z_k \in \mathbb{N}_0$ with $z_i < z_j$ for $i < j$
Output: List of sets of circles $C_1, \ldots, C_k$ with $c \in C_i$ having attributes x, y, r and n

```
result ← empty list
circles ← empty list
for p ← P do
    APPEND(circles, MAKECIRCLE(point.x, point.y, r_min, 1))
for i = k, ..., 1 do
    mapWidth ← 256 · 2^{z_i}
    resolution ← B_crs.width / mapWidth
    B ← PROJECTBOUNDS(B_crs, resolution)
    circles ← PROJECTCIRCLES(circles, resolution)
    C_i ← CMQ˜(circles, r_min, δ, B, r_max)
    circles ← C_i
    APPEND(result, C_i)
return result    // {C_k, ..., C_1}
```

5.5. Generation of Multiple Zoom Levels

In an interactive map application, users explore the aggregated point set. They zoom into interesting areas and zoom out to get a better overview of a larger area. These zooming actions invalidate the initial visual point clustering because the projection of the coordinates of the map extent to pixel values changes. Therefore, it is necessary to calculate a new set of circles. An application might try to calculate several zoom levels in advance to achieve a better responsiveness for the user as a form of prefetching.

For *CMQ*, it would thus be advantageous to speed up the calculation of multiple zoom levels. Fortunately, we can achieve this by reusing results. More precisely, we use the output of an aggregation of zoom level z as input for any aggregation with zoom level z_{new}, with $z_{new} < z$. In particular, we expect zoom level z_{new} to be $z - 1$. We therefore reduce the size of the input for a subsequent zoom

level from n (the number of points) to m (the number of resulting circles from the previous zoom level), which is especially advantageous if $m \ll n$. This incremental calculation is possible because an overlap at a higher zoom level would also occur at a lower zoom level. This monotonicity is true as long as the circles do not become smaller after merging. Hence, reusing the previous results as input makes it possible to avoid redundant computations. Jänicke et al. [JHS13] also made use of monotonicity for computing their multi-level aggregation. In their case, they iteratively remove edges in a Delaunay triangulation for subsequent zoom levels.

Algorithm 5.7 shows the pseudocode for the computation of multiple zoom levels. It uses the same input as Algorithm 5.4, but has a set of k zoom levels $z_1, \ldots, z_k$, with $z_i < z_j$ for $i < j$, as additional input. After initializing the result list (line 1), the algorithm creates for each point one circle using the minimum radius r_{min} (lines 2-4). Starting from the highest zoom level (line 5), it calculates the map bounds and the resolution (line 6 and 7) and maps the data into the current zoom level (line 8 and 9). The algorithm then calls *CMQ* (cf. Algorithm 5.4) for generating the circles for the current zoom level (line 10). These circles then represent the input for the next lower zoom level in the next iteration (line 11) as well as one result entry (line 12). Note that we again use the slightly modified version CMQ˜ of Algorithm 5.4, which accepts circles as input instead of points and skips the calls to makeCircle. This corresponds to the definition in Section 5.4.

5.6. Experiments

In this section, we compare the *Circle Merging Quadtree* (*CMQ*) to the other suitable methods discussed in Section 3.3. These are

- the Delaunay-based aggregation method of Jänicke et. al [Jän+12] (*GeoTemCo*) and
- the quadtree-based approach of Bereuter et. al [BW13] (*quadtree*).

In addition, we apply our grid-based preprocessing step to *CMQ* and specify the memory type (*vector* for array and *hash* for hash map) as well as the space-filling curve type (*hilbert*, *snake*, *z* and *xy*, cf. Section 3.3). For example, CMQ_{plain} stands for *CMQ* without preprocessing and CMQ_{hash_xy} for *CMQ* with preprocessing based on a hash map and the XY curve.

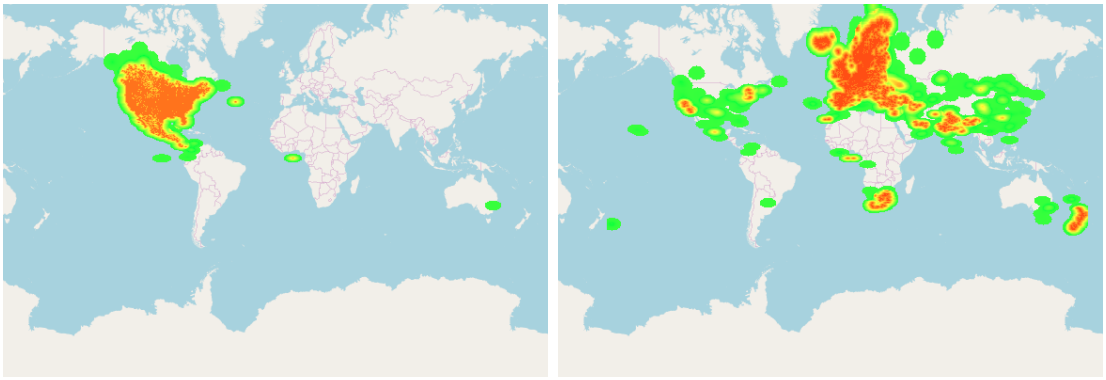


Figure 5.6.: Two example data sets: the left side shows the distribution of 910 784 occurrence points of cooper's hawk in North America and the right side shows the distribution of 768 436 occurrence points of the greylag goose which are spread across the world.

For all computations we set the minimum radius $r_{min} = 2.5\,\text{px}$ (diameter of 5 px) and the inter-circle distance $\delta = 1\,\text{px}$. For specifying the maximum radius we have chosen for comparability that all methods use the rule of thumb of Jänicke et al. [Jän+12] of $r_{max} = 4 \cdot log_2(n)\,\text{px}$. This corresponds to the transformation function Log_2 in Section 4.5.

We implemented all methods in Rust[1] version 1.24. The experiments have been performed on an Intel i7-3770 CPU running at 3.40 GHz and having 24 GB of RAM. The comparison of methods considers the runtime as well as the quality. For *CMQ* we additionally investigate the stability and compression performance.

For this evaluation we extracted 50 real data sets from GBIF[2], which represent the (worldwide) locations of observations of different species (cf. Section 1.2). Their sizes vary from several hundred to several million data points. We obtained these data sets by partitioning all observation data by species name and choosing partitions in a way that favors larger cardinalities. We implemented this by ranking the partitions in ascending order of their cardinality and using rank proportionate sampling (roulette wheel selection [LL12]) such that they were chosen with a probability of $\frac{rank_i}{\sum_i rank}$. These data sets contain different data patterns, i.e., more or less dense regions, larger and smaller distributions of data and differently shaped hot spots. Figure 5.6 shows heat maps of two exemplary data sets to emphasize this. While the left-hand side shows 910 784 occurrence points of

[1] www.rust-lang.org
[2] Global Biodiversity Information Facility, www.gbif.org

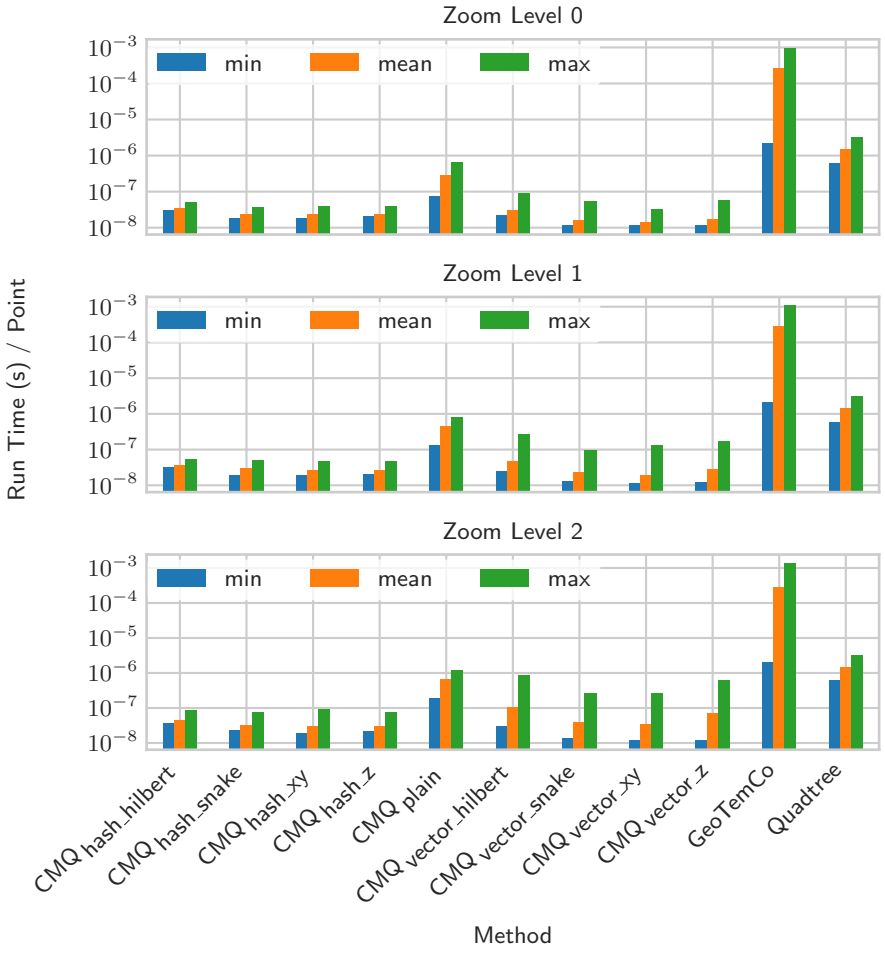


Figure 5.7.: The minimum, average and maximum runtime per point for the 11 algorithms.

cooper's hawk in North America only, the right-hand side shows 768 436 occurrence points of the greylag goose that are more evenly distributed across the world. We have made all data records available under the URL dbs.mathematik.uni-marburg.de/downloads/data/an_efficient_algorithm_for_visual_clustering/, which allows the reuse of the data and reproducibility of the results.

5.6.1. Runtime

For analyzing the runtime, we measured the wall-clock time of the execution. Note that we excluded the initial I/O-time for loading the data sets into main

memory. This is reasonable because it is exactly the same operation for each algorithm.

We consider two different scenarios for the performance evaluation. The first one focuses on a single target zoom level. This is particularly important for exploratory data visualization, where immediate response times are crucial for an initial visualization. The second one considers the runtime for computing multiple zoom levels at once. This is applicable when users want to investigate a single data set in more detail (cf. Section 5.5). For both scenarios, we performed the computations on all 50 data sets and recorded the individual runtimes. We repeated each experiment five times and report the averaged results.

For the first scenario, we measure each zoom level individually and report the results for the different methods in Figure 5.7. Here, we display the minimum, median and maximum runtime per point. We have calculated these times by dividing the total runtime for a data set by the number of points in that data set. This allows us to compare the runtimes independent of the size of the data set. Note that we used a logarithmic scale for the y-axis because of the large differences in the runtimes of the methods. It is noticeable that the *Circle Merging Quadtree* outperforms the other two methods at least by an order of magnitude. The plots of the results for three zoom levels make it apparent that all methods are quite stable and slow down only slightly for larger zoom levels. It is also evident that *CMQ* with preprocessing has an order of magnitude lower runtime than *CMQ* without preprocessing. In addition to providing a stable algorithm, it seems to be highly advantageous for the runtime. The hash map storage has a better maximum runtime in comparison to the array storage. Furthermore, the snake and Z curve seem to have a slightly better performance than the Hilbert and xy curves.

For further investigations of the behavior of *CMQ*'s preprocessing implementations, we present another view on the data. The boxplot in Figure 5.8 shows the differences between hash map and array storage. Array storage is more efficient for small zoom levels, while hash map storage is more efficient for larger zoom levels. This can be explained by the increasing sparsity of the map as the map becomes larger. Here, the array becomes significantly larger than the hash table, while the number of occupied positions in the array, which are the actually used grid cells, becomes quite small.

Figure 5.9 shows the runtimes per point for the 50 data sets for CMQ_{hash_z} in more detail (in this example for zoom level 3). Since the values are not aggregated, we can assess the differences between smaller and larger data sets. We can notice an overhead for smaller data sets and a consistent runtime per point for larger data

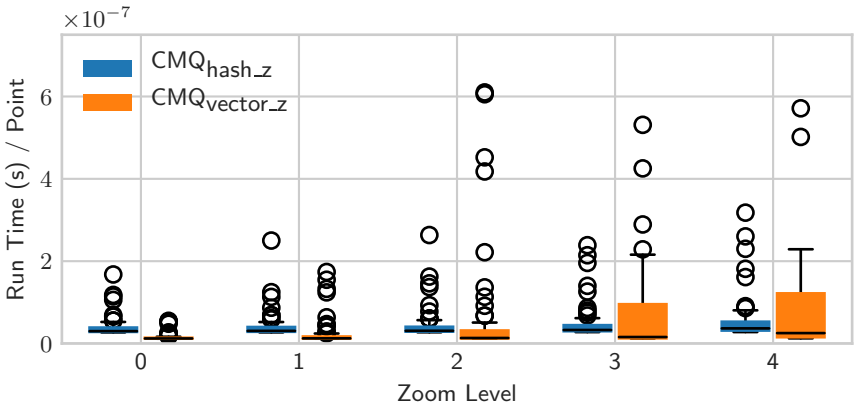


Figure 5.8.: The runtime per point for CMQ_{hash_z} and CMQ_{vector_z}.

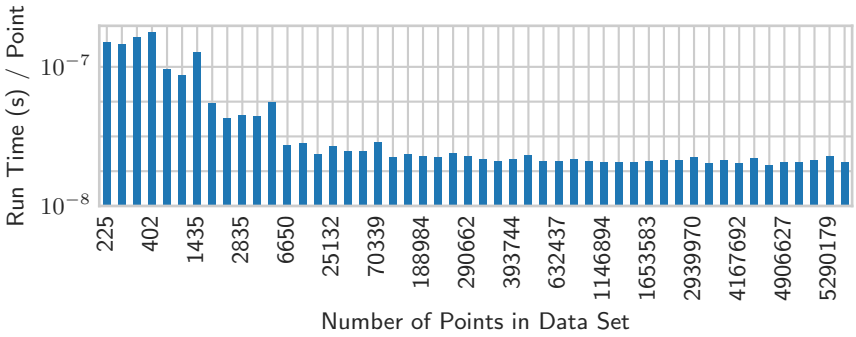


Figure 5.9.: This figure shows the runtime per point of CMQ_{hash_z} for the 50 data sets. The x-axis indicates the number of points in the data set.

sets. This overhead for smaller data sets is a factor to explain the outlier values in Figure 5.8. The consistency on the right side of the plot confirms the linear scaling of the runtime for larger cardinalities of the input data.

For the second scenario, we report the computation times for a total of five zoom levels. All methods can exploit the fact that multiple zoom levels are computed at once. *CMQ* uses the method described in Section 5.5. *Quadtree* takes advantage of creating the data structure only once and traversing it by outputting the nodes that depend on the minimum circle size for that zoom level (which depends on the transformation from map units to pixel values). *GeoTemCo* exploits the hierarchical clustering strategy and can output the result of a single zoom level as an intermediate step. Figure 5.10 shows similar results for all data sets (sorted wrt. their size) in comparison to the first scenario where only one zoom level is considered. *CMQ* also performs better than the other two methods here, in par-

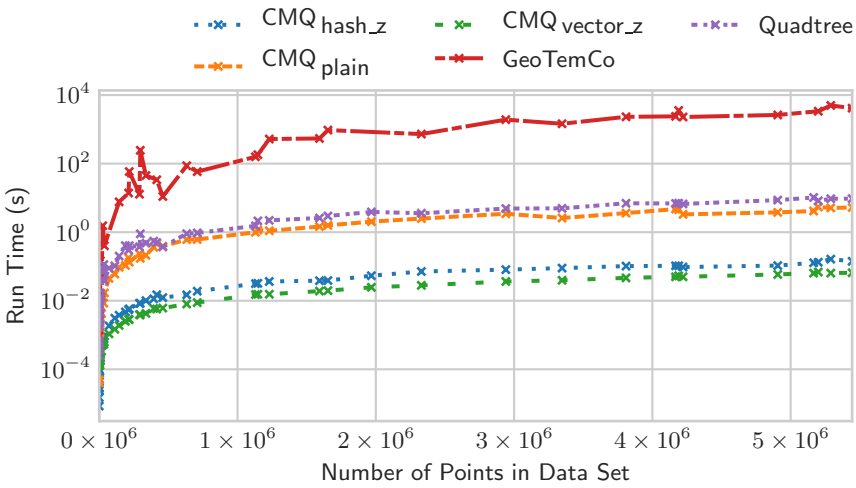


Figure 5.10.: The runtime of five algorithms for five zoom levels over the number of points of the 50 data sets.

ticular with preprocessing. Note that we again use a logarithmic scale for the y-axis. Moreover, there is a growing divergence for the larger data sets between *CMQ* and *GeoTemCo*. This is an indicator for the superior scalability of the method.

Figure 5.11 depicts the runtimes of the different preprocessing parametrizations of *CMQ* as a function of the 50 data sets (sorted by size). Note that we again computed all five zoom levels. While the hash map showed a better scalability in the first scenario, the array storage has a better overall runtime for multiple zoom levels here. The Z curve seems to be the space-filling curve that poses the least overhead for the computations. The Hilbert curve, on the other hand, performs worse regardless of the storage type.

5.6.2. Quality of Results

To evaluate the visualization quality we use the seven quality criteria introduced in Chapter 4. We apply all algorithms to five zoom levels for each of the 50 data sets and obtain 250 results per method. The results are summarized in seven boxplots in Figure 5.12, which displays an individual boxplot for each method and each criterion. Recall that the quality measures pose utility values. A value of one indicates the best result regarding a certain criterion, whereas zero is the worst result.

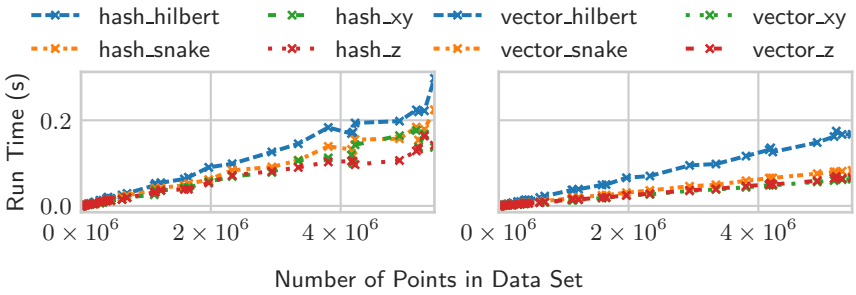


Figure 5.11.: This figure shows the runtime of all *CMQ* algorithms for five zoom levels over the number of points of the 50 data sets. This plot compares the runtimes for *CMQ* with hashing (left) and array storage (right).

The boxplots for the criteria *centered* and *distance* show no difference among the methods. All of them produce very good results. The criterion *overlap* is completely fulfilled by all variants of the method *CMQ* as well as *GeoTemCo*. The *quadtree* produces, however, very poor results. This is due to the circle construction, which determines the radius by the number of points in a subtree. If this subtree contains a lot of points, the resulting circle greatly exceeds the bounding box of the node, which can lead to an overlap with circles of other nodes. All methods produce a small number of points that are not located within any circle (*unassigned*). Within the circles, the points follow a uniform distribution to a certain degree. However, none of the four methods is producing circle representations with good results for this criterion. It seems that this criterion is difficult to satisfy because not all point distributions can be perfectly approximated by non-overlapping circles. Even though we do not obtain optimal results for *zoom* consistency, the results obtained from *CMQ* and its variants as well as *GeoTemCo* are acceptable. The *quadtree* approach is significantly worse in this regard. When zooming in, it is possible that a deeper tree level is considered for the creation of the circles. This means that a circle is split up into (up to) four circles that are located at the centers of the child nodes. The center of the parent node may thus be potentially empty on the map. Finally, the criterion *area* varies considerably between all data sets, at least if *CMQ*, *GeoTemCo* or *quadtree* are applied. For some data sets, we obtain satisfactory results, whereas for the majority of data sets the results are very poor. One explanation is the incorporation of the minimum radius r_{min}, which does not allow a proportional scaling that would undercut this threshold.

In our next evaluation, we give all individual quality criteria equal weight and sum them up to get a scalar quality criterion for each method on each data set.

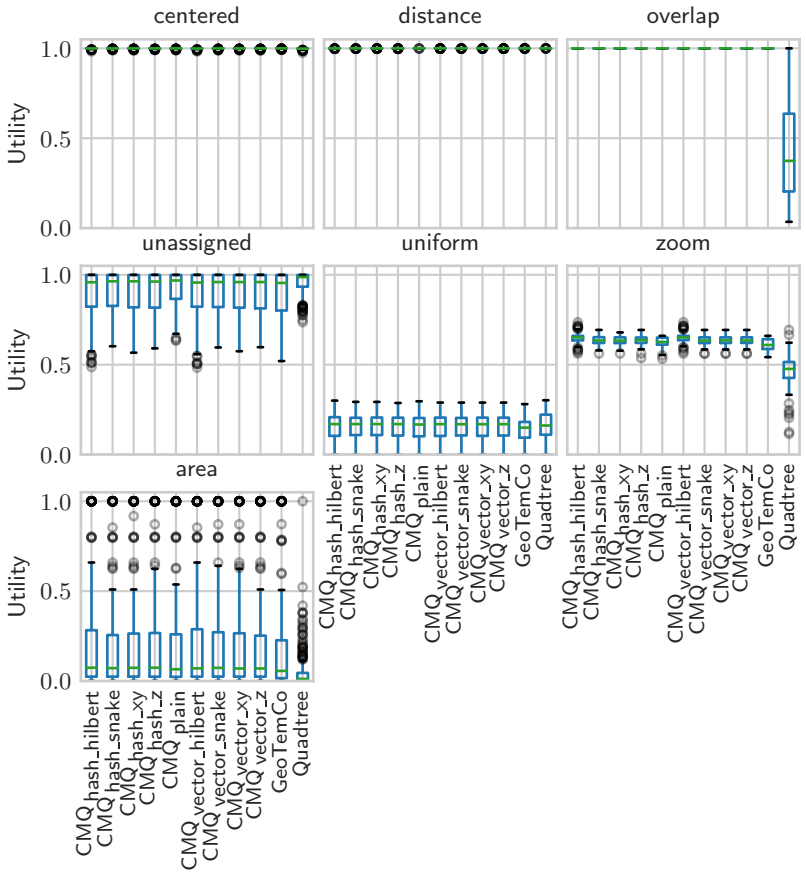


Figure 5.12.: These boxplots show the performance of all methods regarding the seven quality criteria individually.

The results for this combined criterion are depicted again in the form of boxplots in Figure 5.13. As can be recognized, the *CMQ* variants lead to the best overall results, closely followed by *GeoTemCo*. *Quadtree* performs clearly worse than the other methods. In order to confirm this visual impression we also performed several Wilcoxon tests [CF14]. In contrast to the popular t-test, it does not require a uniform distribution of the data. More specifically, we can reject the null hypotheses that the pairwise differences of the quality values between each *CMQ* variant and *GeoTemCo* or *Quadtree* symmetrically distribute around 0 with over 99 % confidence. This is in accordance with the visual impression of the boxplots.

Figure 5.14 shows the results of the visualization of the occurrence records of the Central European wolf spider (Piratula hygrophila), which has most records in Belgium. We can observe that *CMQ* provides a very clear and informative result

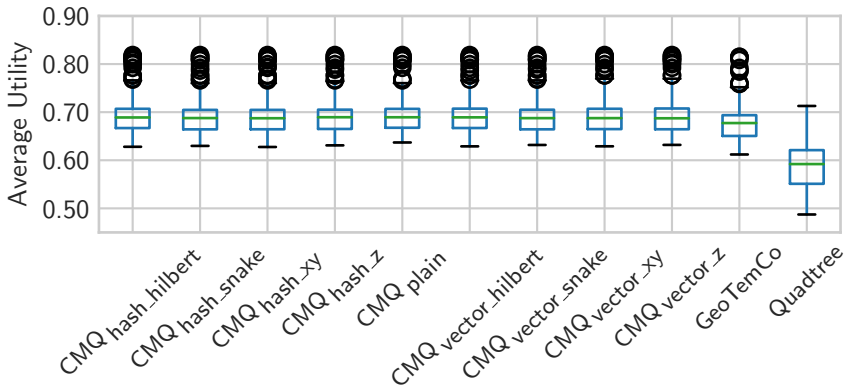


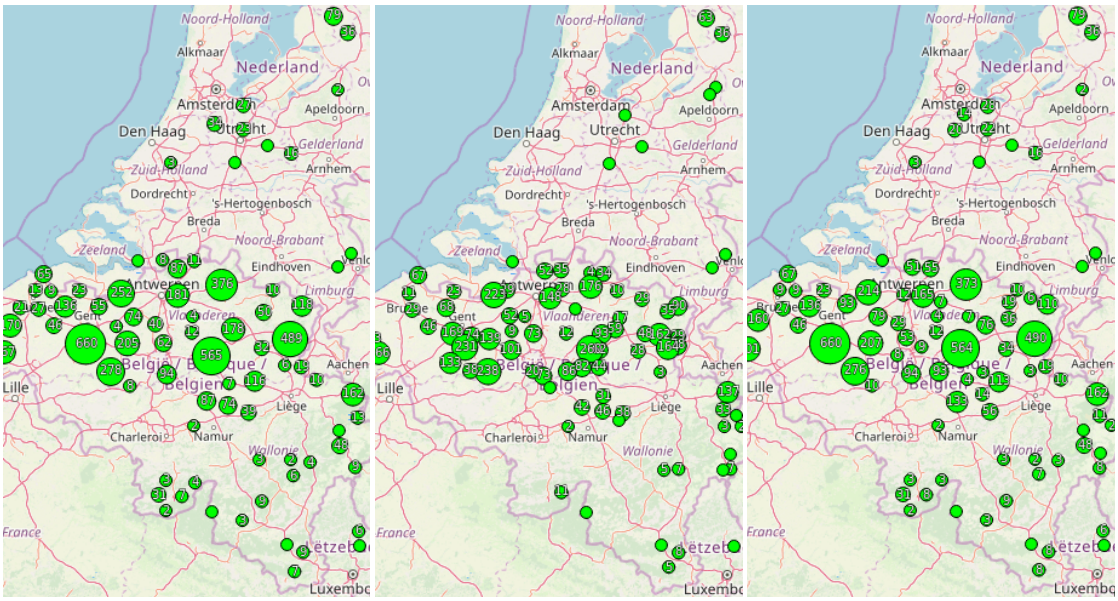
Figure 5.13.: This boxplot shows the performance of all methods as an equally weighted average of all seven quality criteria.

(cf. Figure 5.14 (a)). On the other hand, *Quadtree* (cf. Figure 5.14 (b)) produces several overlaps that make it more difficult to recognize and distinguish between individual clusters. The result of *GeoTemCo* (cf. Figure 5.14 (c)) is very similar to the result of *CMQ*, which is as expected.

5.6.3. Stability

In Section 5.4 we discussed the stability of the results of *CMQ* with respect to the insertion order of the input data. In this subsection, we evaluate how stable *CMQ* is without preprocessing. Furthermore, we validate that the preprocessing step indeed solves the stability problem. In order to measure stability, we use the intersection over union (IoU) of the areas between two result sets. This is calculated by using a Monte Carlo simulation and by calculating the difference between the number of commonly shared points and the number of the points that fall in one of the resulting sets of circles. We created a first assessment by generating 8 permutations from each of the 50 data sets and comparing these permutations with each other. This leads to a total of $50 \cdot \binom{8}{2} = 1\,400$ comparisons. As a second assessment, we use the native and reverse order of the data to investigate probable worst-case scenarios. This leads to another 50 comparisons.

Figure 5.15 shows the results of the stability experiment. At first, we see that results of *CMQ* without using the preprocessing are, as assumed, not perfect, but nevertheless of high stability. The median is close to 95 %, but there are also some results below 75 %. The vector approach, which uses both the one-dimensional array and the hash map with the sorted overflow lists, produces 100 % stability



(a) (b) (c)

Figure 5.14.: This figure shows the three results of the methods (a) *CMQ*, (b) *Quadtree* and (c) *GeoTemCo* for the Central European wolf spider (Piratula hygrophila).

for all data sets. In conclusion, the preprocessing step leads in each case to stable results.

5.6.4. Compression

In addition to assessing the qualitative aspects of point aggregations, it is interesting to look at the advantages regarding mobile applications that suffer from low internet bandwidth. This means that any form of data compression in this domain is highly advantageous.

For this experiment, we again use all 50 data sets. Figure 5.16 shows the compression rates for different zoom levels by using the *Circle Merging Quadtree*. They are calculated by dividing the byte size of the input data set of points by the byte size of the output data set of circles (thus also considering the additional radius field). Note that we used the *plain* version of *CMQ* without preprocessing because it does not lead to significantly different results. Since the visual space is limited, and we assess the performance using large data sets with millions of points, the resulting

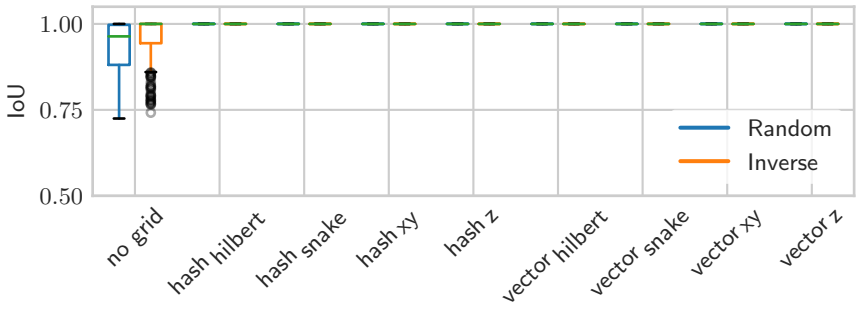


Figure 5.15.: This boxplot shows the intersection over union (IoU) for the clustering outputs of the different *CMQ* methods. Inputs are either random permutations or inverse data sets.

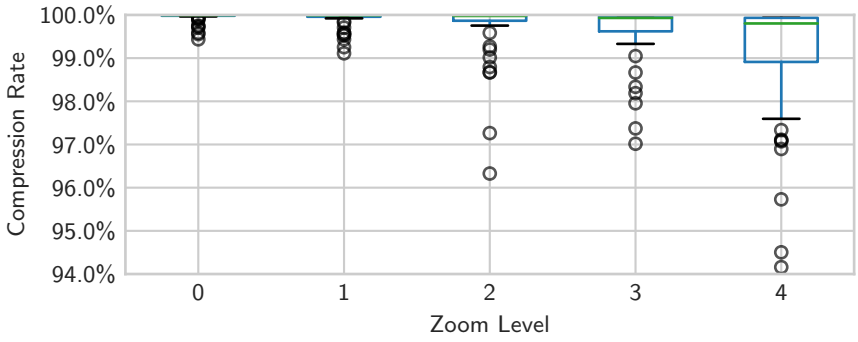


Figure 5.16.: This boxplot shows the compression rates of *CMQ* for different zoom levels. Higher zoom levels lead to more space and, hence, to fewer overlaps.

numbers of circles that range from several dozens to a few thousands yield compression rates above 98 %. This is in accordance with our predicted bound of the maximal number of circles in the output in Section 5.2.

5.7. Summary

This chapter presented *CMQ*, the *Circle Merging Quadtree*, which is a new method for aggregating point data sets into a set of circles. Thus, it represents a visual point clustering algorithm. We provided implementation details as well as a theoretical analysis of runtime and space complexity. In addition, we discussed the concept of stability and presented a preprocessing step that poses a solution for a stable *CMQ* variant by using a spatial grid. Furthermore, we provided a *CMQ*

variant for calculating multiple zoom levels at once by exploiting monotonicity of the computational results.

The experiments confirmed the theoretical analysis that *CMQ* is a suitable visual point clustering algorithm for big data scenarios. In particular, *CMQ* with preprocessing is much faster than the plain *CMQ* variant, although it was not the intended feature of the design considerations. Besides outperforming competing algorithms, *CMQ* also provides qualitatively good results that are at least as good as the results from the competitors. The stability evaluation verified the theoretical proof that the preprocessing does indeed lead to stable results. Preprocessing is therefore advantageous in any case since it does not have any downsides, in particular none for the quality criteria. Finally, the compression rates of *CMQ* are, as expected, very high. In accordance with the assumption that the number of circles is very small in contrast to the number of points from the input ($m \ll n$), it revealed one of the general strengths of visual point clustering. It is a win-win situation that a decrease of necessary space is accompanied by an increase in information about the data for the user.

6
CMQ Extensions

This chapter presents extensions of the *CMQ* algorithm. Here, *CMQ* is considered as the basic version upon the following methods build up independently. The extensions allow *CMQ* to be more powerful by adding additional features. Furthermore, each extension must adhere to the linear runtime of the algorithm such that it stays relevant for big data scenarios. The chapter starts with Section 6.1 and presents an extended version of *CMQ*, which aims to provide summaries of miscellaneous attributes. These attributes have non-spatial characteristics. The section provides means for numerical as well as textual attributes. Then, the chapter presents methods to cope with multiple data sets at once or multiple classes within the data. Section 6.2 presents two approaches. First, it presents pie chart maps. Second, and in more detail, it presents circle packed maps. For this method, the section additionally presents experiments and a user survey regarding runtime and quality. Finally, Section 6.3 provides a brief summary of this chapter.

6.1. Summary of Miscellaneous Attributes

This section presents extensions to the *CMQ* algorithm to deal with miscellaneous attributes of the input data set. In the previous chapter, we only considered the spatial attributes of the data set. However, spatial data typically has further attributes in real-world scenarios. In our visual point clustering scenario, we cannot always incorporate these values into our visualization. Section 6.2 will present means for this special case. Generally, we can present these values alongside with the visualization, e.g. in a data table or in additional plots. Since we have to output attribute information for every circle (i.e. cluster), we have to provide aggregated information here as well. Hence, we discuss suitable aggregation methods that we can utilize for the miscellaneous attributes in the spatial clustering process. The resulting values provide characteristics for the data that is represented by

the circles. In the following, we will distinguish between numerical and textual attributes.

6.1.1. Numerical Attributes

Aggregating numerical attributes is basically the calculation of statistics on the underlying data. This means we fall back on computing basic characteristics with standard algorithms. However, we have to consider two restrictions: First, *CMQ* runs in linear time. In order to preserve this runtime, we have to use equally efficient algorithms. Second, the clustering process merges data sets iteratively. Hence, an incremental algorithm to generate statistics is required. Note that statistics like median are known as examples that are expensive to compute in an iterative fashion. However, incremental statistics are not strictly required since we can use a fallback. We first store all attribute data in (linked) lists that we merge in the circle merge step. Then, we calculate the statistics as a last step. This does, however, increase the space complexity of *CMQ* to $O(n)$.

Many essential statistics can be calculated in a single pass over the data set. This means that it is possible to calculate them in linear time, so that the runtime of *CMQ* is preserved. In the following, we discuss a selection of them. We use X for denoting a list of n numerical values $(x_1, \ldots, x_n)$.

Average/Mean To calculate the average of an attribute of a data set in a single pass, one can basically add up all values and divide the sum by the number of values. An incremental approach [Fin09] is the following:

$$\overline{x}_n = \frac{(n-1) \cdot \overline{x}_{n-1} + x_n}{n} = \overline{x}_{n-1} + \frac{x_n - \overline{x}_{n-1}}{n} \ .$$

For updating the mean after merges, it is also possible to use this formula for merges of two sets of values, in which p is a value between 1 and n:

$$\overline{X} = \overline{X_{1,\ldots,p} \cup X_{p+1,\ldots,n}} = \frac{p}{n}\overline{x}_{1,\ldots,p} + \frac{n-p}{n}\overline{x}_{k+1,\ldots,n} \ .$$

Variance and Standard Deviation To calculate the variance of an attribute of a data set in a single pass, one can generally exploit the formula

$$Var(X) = E(X^2) - E(X)^2 \ .$$

However, it can lead to arithmetic overflows for large data sets if the addition of squared values leads to overflows of the numeric variable. In order to save

space in the *CMQ* computation, it is not always the best option to simply use more bits for storage. Welford [Wel62] provided the following algorithm as a stable online variance calculation:

$$\begin{aligned} \delta &= x_n - \overline{x}_{n-1} \\ M_{2,n} &= M_{2,n-1} + \delta(x_n - \overline{x}_n) \\ s_n^2 &= \tfrac{M_{2,n}}{n-1} \ . \end{aligned}$$

Here, s_n^2 denotes the (corrected) sample variance for n values. $M_{2,n}$ represents the square of differences to the mean. Since the δ values generally are much smaller than the values themselves, the additions are much more robust against arithmetic overflow errors.

Skew & Kurtosis The skew and kurtosis values of a list of values can be calculated similarly to the variance algorithm of Welford. Pébay [Péb08] introduced the calculation of values for $M_{3,n}$ and $M_{4,n}$:

$$\begin{aligned} s_n^2 &= s_{n-1}^2 + \tfrac{\delta}{n} \\ M_{3,n} &= M_{3,n-1} + \delta^3 \tfrac{(n-1)(n-2)}{n^2} - \tfrac{3\delta M_{2,n}}{n} \\ M_{4,n} &= M_{4,n-1} + \tfrac{\delta^4(n-1)(n^2-3n+3)}{n^3} + \tfrac{6\delta^2 M_{2,n}}{n^2} - \tfrac{4\delta M_{3,n}}{n} \ . \end{aligned}$$

This enables incremental updates without introducing probable arithmetic overflows.

Min/Max The calculation of minimum and maximum are trivial. It suffices to introduce a variable that stores the intermediate min/max value and update it during the merge step.

Top k In order to store more than one maximum value, we have to introduce a data structure that replaces the single numeric variable. We omit the least k discussion here since it is almost identical and just requires switching the comparison function. The data structure is a bounded min heap that stores k elements at maximum. Frederickson [Péb08] introduced this selection algorithm. At first, it fills the heap until it contains k elements. Then, it can compare the new value to the top value of the heap, which is the smallest one of the k elements. Whenever a new, larger value occurs, it swaps the top value of the heap with this value. Then, reorganizing the heap leads to the smallest value being on top again. Since the heap is limited to k elements, we can consider the runtime to update it to be constant. In total, this leads to a linear runtime.

The calculation of the median as well as the mode of a list of values is not computable in a single pass with only constant memory [MP80]. For the median,

the standard algorithm sorts the data and takes the value in the middle (or an aggregate of the two middle values for an even number of data values). Since this requires $O(n \cdot log(n))$ time, it is necessary to use selection algorithms instead [Cor+09]. For instance, the *median of medians* algorithm provides a pivot element that is very close to the real median. Actually, practical applications usually use approximate algorithms. They run in $O(n)$ time or $O(n)$ expected time, respectively.

We treat temporal data attributes in the same manner as numeric values. Here, UNIX timestamps (seconds, optionally with fractions and minus leap seconds, since Thursday, 1 January 1970) are a common numerical representation. Viable statistics are the time span ($max-min$) and the standard deviation.

As stated before, we usually display these statistics in plots or data tables. Additionally, it is also possible to use shadings of a color to indicate differences. For example, a light blue color indicates older mean values and stronger blue colors imply that there are more recent mean values in a circle.

6.1.2. Textual Attributes

Textual attributes have different properties than numerical attributes. First of all, the individual attribute values are of different or unknown sizes. A dictionary encoding (also called dictionary compression or domain encoding) [WMB99] can help to solve this problem. It leads to the storage of numerical identifiers instead of strings and allows looking up the strings whenever necessary. Furthermore, it is difficult to aggregate textual attributes. Since these values are of nominal (or categorical) type, they do not support certain operations (additions, multiplications) that are essential for numerical values. For instance, most of the time it is rather useless to calculate the average string or the standard character deviation for a list of scientific species names. Instead, it is meaningful to provide a list of characteristic values. This is similar to the top k results of the previous subsection.

We provide three methods to generate such a list of k representative textual values. First, we can use sampling to obtain this list. Reservoir sampling (cf. Section 3.3) provides a good way to incrementally compute it. We first collect k items in a list and swap each new value with one of these items with a probability of k/i. Here, i is the running index of the values in a loop. Note that in practice it is only necessary to calculate the index of the next value that leads to a swap and ignore all values in between.

Second, we can retain the values that are closest to their circle center. By doing so, we expect these values to be of high relevance to the user's viewpoint. For this method, we have to adapt the merge method of two circles. Here, we keep k elements and discard the rest based on the distances to the (new) circle center. As data structure we employ a max heap for the merge step or simply lists for small k. Since k is a constant, the overhead is constant as well.

Third, we can compute the values that occur with the highest frequency. Here, we introduce a hash map for every circle. The key is the index of the encoded dictionary and the value is a count variable that is incremented on every occurrence of a value. Then, we find the most frequent k values by using the min-heap technique as explained in the previous subsection. For merges, we have to update the hash map of the first circle with the key-value pairs of the other circles, each time using a traversal of the hash table. To avoid unnecessary costs we fix the largest hash table and insert the values of the remaining participating circles of the merge. Note that this method requires $O(n)$ additional space for *CMQ*.

Since the total number of values cannot exceed n, the (total) sizes of the hash tables are bounded by $O(n)$. This is the case since the visual point clustering yields a non-overlapping partitioning of the attribute values. Hence, we are still able to compute this in $O(n)$ expected time. This method is especially useful if we work with a constant number of different values, which is the case for classes or categorical attributes.

6.2. Visualizing Multiple Classes and Data Sets

CMQ ensures an efficient visualization of a point data set, while simultaneously complying with a range of visual qualities. For instance, it removes clutter and occlusion by not allowing any overlap. However, when we want to visualize multiple point data sets at once, the same problems arise that were described in Chapter 1. Figure 6.1 shows the combined visualization of three data sets. The displayed species occurrence points show European wildcats (red), European moles (green) and European rabbits (blue) on top of each other. Here, the top layer hides most of the two underlying layers of data. For instance, the species distribution of European wildcats is hardly recognizable in this visualization.

In our setting, the visualization of multiple data sets is equivalent to the visualization of a single data set that contains multiple classes. For instance, a data set in GBIF can contain records based on *literature*, *human observations* or *machine*

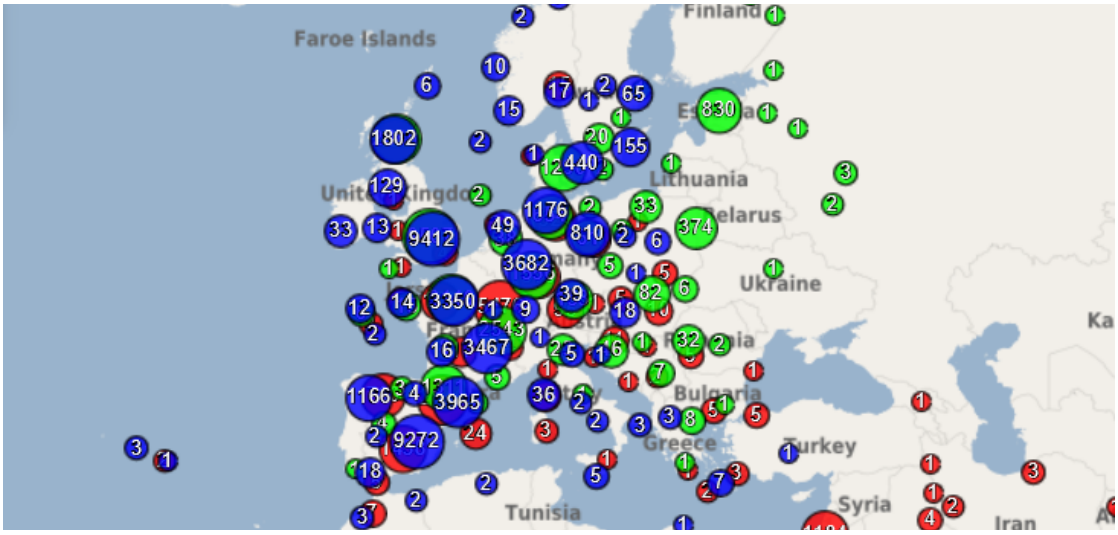


Figure 6.1.: This graphic shows a combined visualization of three data sets that induce an overlap among the circles.

observations. For q different data sets, we can simply merge these records and introduce a new class attribute *cls* with the values $1, \ldots, q$. Therefore, we can treat the data in both scenarios equally. For the sake of simplicity, we consider the class label to be an integer. For textual class labels, we can easily introduce a numeric identifier.

In general, we want to display data from different classes distinctively on the map. In literature, there are generally two options to solve this problem (cf. Section 3.3). First, we can display each circle on the map as a pie chart that visualizes the fractions of the classes. Second, we can display circles of different classes next to each other on the map such that they do not overlap each other. In each case, we give the individual classes distinct colors in order to differentiate them visually. The following subsections detail on both options, but have an emphasis on the latter approach.

6.2.1. Pie Chart Maps

Pie Chart Maps [Slo+09; Gha+14] represent one way to visualize multi-class data with *CMQ*. Figure 6.2 shows an exemplary visualization of a puma data set in North and South America with three classes (blue, green and red) that were derived by correlations with environmental data and partitioning the data via thresholds. In the figure, each circle contains an individual pie chart. The slices

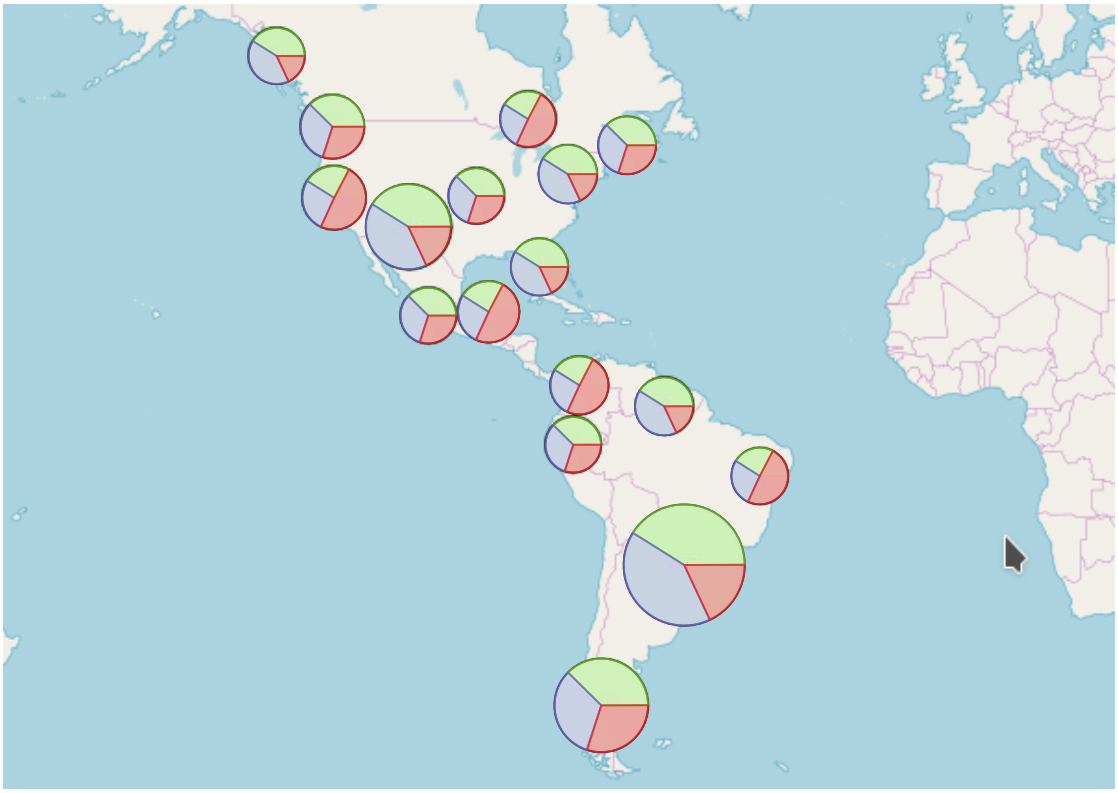


Figure 6.2.: This figure displays a map of differently sized circles. Each of them shows different class occurrences in the form of a pie chart.

correspond to the fractions (count) of the classes in the data set that are assigned to the corresponding circle. This visualization form has advantages and disadvantages.

It is positive that it is straightforward to extend the *CMQ* algorithm to implement this approach. Here, it is only necessary to introduce an individual hash map for each circle in which the algorithm can store the number of class occurrences. During the merge step, the algorithm has to add the hash map values of all but one circle to the values of the remaining circle. This yields the class counts for the new, merged circle. This technique is equal to the counting of classes presented in Section 6.1. Besides that, pie chart maps are expressive and understandable because they are a fundamental and ubiquitous technique for data visualization [Slo+09; Mun14; HKP11]. They allow to exactly see the class differences within the circle and still present the total differences in the number of points due to the proportional circle sizes.

A negative aspect is that the circle fractions are hardly intercomparable with

fractions of the other circles. Thus, it is complicated to assess the differences of a single class in circle A to the same class in circle B. Furthermore, pie charts within the minimum circle (with radius r_{min}) are generally hard to recognize. It is therefore necessary to increase the minimum radius accordingly, which leads to more overlap, and thus to a coarser visualization.

For client-side visualization, the algorithm has to attach to each circle an additional set of classes with their counts or fractions. Then, map libraries like OpenLayers [GSH15] allow the incorporation of visualization libraries like d3 [BOH11] (cf. the OpenLayers d3 integration example[1]). Since the number of outputted circles m is quite small, the overhead of drawing multiple small charts on the map is moderate.

6.2.2. Circle Packing

Drawing circles of different classes on the map is an alternative to pie chart maps. For this approach, we apply the same visual quality aspects to this advanced multi-class visualization as to the single class problem. Thus, it is of utmost importance to still disallow any overlap. For *CMQ* it is only valid to merge circles that are of the same class. But if the algorithm detects an overlap of two circles of different classes, it is not clear how it should cope with it.

A general option is to displace the circles such that they do not overlap. However, this does not only introduce an error in quality that occurs since the circles move away from the underlying attached points. In addition, the constraint map size does not allow this option. The following example underlines this statement. Assume the map is tightly filled with circles of class A such that it is not possible to move one circle without causing any overlap. Then, if any circle of class B should be introduced into the visualization, at least one circle of class A must disappear or fall out of the map when a displacement happens.

Jänicke et al. [Jän+12] developed a method that we want to adapt for *CMQ* in the following. Here, we compute the set of circles as in the pie chart method. But instead of creating the pie charts, we introduce another step of packing a new set of circles of the individual classes into the *overall* circles. We call these *overall* circles placeholder circles. In the following subsection, we will describe them in more detail.

An advantage of the circle packing approach is that this allows a comparison of circles belonging to different classes as long as they are proportionally scaled

[1] openlayers.org/en/latest/examples/d3.html (accessed November 10, 2018)

depending on their attached number of points. It is easy to distinguish different circles and assess their sizes not only locally, but also globally. Furthermore, we will show a way how to integrate this packing into the *CMQ* algorithm without having to make fundamental changes.

A disadvantage is that this method is more complex than calculating the pie chart fractions. It requires an additional step of circle packing. Furthermore, having circles of multiple classes on the map means less space for the individual classes. In more detail, if a circle of class A covers a region, e.g. Germany, it is not possible for a circle of class B to cover the same region. This inevitably leads to less covered space. We will investigate this in more detail by applying the quality measures in the experiment subsection.

Algorithms

In order to implement the updated merge method, we introduce two new data structures that are necessary to cope with circles of multiple classes. The first one is the extended circle $c^+ := (x, y, r, n, cls, x_d, y_d)$. The first four parameters are equal to the parameters of the original circle definition in Chapter 2. In addition, it contains a class label cls and display coordinates x_d and y_d. The former circle coordinates x and y define the centroid of the data points of the represented data. However, due to the display of multiple classes within the placeholder circle, it is necessary to displace the circle positions such that they can be visualized side by side without overlaps on the map, again. Thus, the display coordinates x_d and y_d can differ from the data coordinates, which refer to the centroid of the points in the cluster. The differentiation of these two coordinate pairs is very useful for optimizing the packing of the extended circles within their placeholder circles. As a reminder, for the sake of simplicity, we consider all class labels cls to be integers.

The second data structure is the aforementioned placeholder circle $c^\circ := (x, y, r, l = \{c_1^+, \ldots, c_q^+\})$. It contains a circle with center and radius as well as a set l of extended circles. For each class, there is at most one extended circle. We consider the extended circles for classes that are not reflected by a placeholder circle as undefined. Moreover, all extended circles $c_1^+, \ldots, c_q^+$ lie inside the placeholder circle using their display coordinates.

Algorithm 6.1 shows the algorithm of the extended merge method. Here, in line 1, the algorithm initializes the set of circles that at some point will form the new set l for the placeholder circle. In line 2, the algorithm initializes a map that holds up to q sets of extended circles. In lines 3 to 5, the algorithm iterates over all placeholder

Algorithm 6.1 Function MERGE$_{\text{MULTI}}$ for merging circles with circle packing

Input: Set of placeholder circles $\{c_1^\circ, \ldots, c_k^\circ\}$ with attributes x, y, r and l
Minimum circle radius r_{min} in px
Maximum circle radius r_{max} in px
The total number of points n
The number of classes q

Output: A new, merged placeholder circle c° with attributes x, y, r and l

```
1: mergedCircles ← ∅
2: circles ← empty map of size q from cls to a set of extended circles
3: for c° ∈ {c°_1, ..., c°_k} do
4:     for c+ ∈ c°.l do
5:         circles[c+.cls] ← circles[c+.cls] ∪ c+
6: for i ∈ 1, ..., q do
7:     c+ ← MERGE(circles[i], r_min, r_max, n)
8:     if c+.n > 0 then
9:         mergedCircles ← mergedCircles ∪ c+
10: l ← CIRCLEPACKING(mergedCircles)
11: (x, y, r) ← BOUNDINGCIRCLE(l)
12: return MAKEPLACEHOLDERCIRCLE(x, y, r, l)
```

circles and their sets of extended circles l, and inserts them into *circles*. In lines 6 to 9, the algorithm merges all circles of the same class and inserts the resulting circle into the *mergedCircles* set as long as they are initialized ($n > 0$ indicates that). In line 10, the `circlePacking` method is called to update the (x_d, y_d) coordinates of the extended circles. In line 11, the algorithm calculates a new bounding circle for the set of extended circles by calling BOUNDINGCIRCLE on l. Finally, in line 12, the algorithm creates a new placeholder circle with the placeholder coordinates x and y, the placeholder radius r and the set of extended circles l. We now present four different algorithms for the CIRCLEPACKING method.

Basic Packing Basic packing is based on the work of Jänicke et al. In their method, they use pre-defined circle positions[2] for values of q for one to six. Figure 6.3 shows one example with $q = 4$. These illustrated positions are derived from work of Kravitz [Kra67]. In this approach, the largest circle (depending on the number of represented points) is always placed at the upper left position. Since the goal is to find a small placeholder circle, the second largest circle is placed on the diagonal opposite to the first circle. The figure highlights this on the left

[2] cf. hydra.nat.uni-magdeburg.de/packing/cci/d1.html (accessed Nov. 30, 2018)

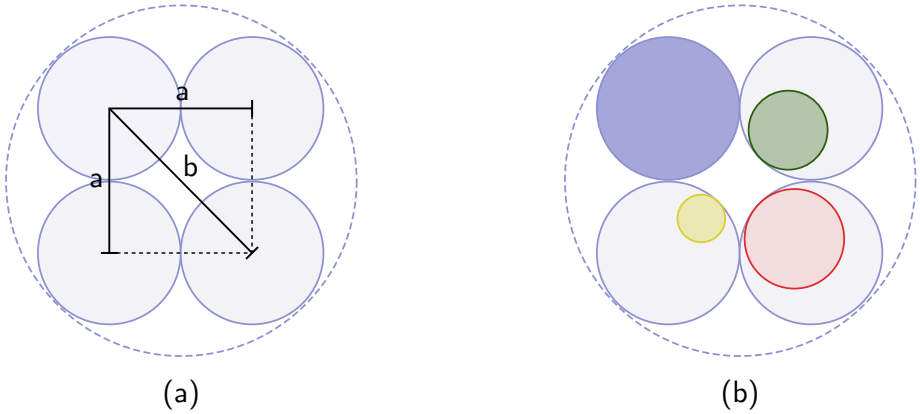


Figure 6.3.: This figure illustrates the packing template (a) and the basic packing algorithm (b) for packing four extended circles $\{c_1^+, \ldots, c_4^+\}$ in one placeholder circle c°.

side (a) by plotting the triangular distances where the hypotenuse b is the largest distance compared to the legs a. Thus, this technique ensures smaller placeholder radii. The two remaining circles are then placed at the two remaining spots. In the following, we refer to the list of q positions for q circles that are packed into a placeholder circle as a *template*. The generation of such a template requires (i) the center of the placeholder circle, (ii) the (largest) radius to use for the q inner circles and (iii) the number of circles q. Furthermore, the right side (b) of Figure 6.3 shows that smaller circles are placed towards the center within their placeholder circles.

Algorithm 6.2 shows the steps of this approach. Here, in line 1, the algorithm sorts the list of circles in descending order by their radius. It can then easily retrieve the largest radius in line 2. In line 3, the algorithm calculates the center of the placeholder circle by calculating the weighted centroid of the circle centers. In line 4, the algorithm creates a template of circle (display) positions as described above for the number of circles $|l|$ and based on the previously calculated center. Then, in lines 5 and 6, the algorithm retrieves the template positions one by one for the circles. It updates the values for x_d and y_d in-place. Finally, in line 7, the algorithm transforms the list of circles into a set that can be used as a new set l for the placeholder circle.

Data-related Packing Data-related packing follows a different approach of computing the template positions. Here, the algorithm tries to place the circles close to their data center. The hypothesis is that this approach increases the quality of the overall clustering. On the other hand, the radii of the bounding circles around the

Algorithm 6.2 Function BASICPACKING for circle packing

Input: Set l of circles $\{c_1^+, \ldots, c_{|l|}^+\}$ with attributes x, y, r, n, cls, x_d and y_d
Output: An updated set of circles $\{c_1^+, \ldots, c_{|l|}^+\}$

1: circles $\leftarrow$ SORT(l, *order by r desc*)
2: $r_{largest} \leftarrow$ circles[0].r
3: $(x_{center}, y_{center}) \leftarrow \left(\frac{\sum_{i=0}^{|l|} \text{circles[i]}.x \cdot \text{circles[i]}.n}{\sum_{i=0}^{|l|} \text{circles[i]}.n}, \frac{\sum_{i=0}^{|l|} \text{circles[i]}.y \cdot \text{circles[i]}.n}{\sum_{i=0}^{|l|} \text{circles[i]}.n} \right)$
4: template $\leftarrow$ CREATETEMPLATE(x_{center}, y_{center}, $r_{largest}$, $|l|$)
5: **for** $i \in |l|$ **do**
6: (circles[i].x_d, circles[i].y_d) $\leftarrow$ GETPOSITION(template, i)
7: **return** TOSET(circles)

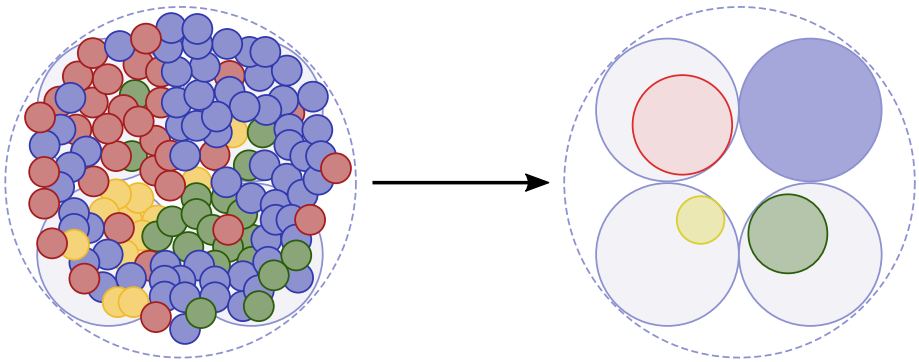


Figure 6.4.: This figure illustrates data-related packing. The circles on the right side are arranged according to the underlying data (left side).

packed (extended) circles are expected to be slightly larger than in the basic packing algorithm. Figure 6.4 illustrates the idea of data-related packing. Here, the circles (right side) are arranged accordingly to the underlying data distribution (left side). For instance, the majority of blue points is at the upper right of the example and the representative circle is in the same position.

Algorithm 6.3 shows the steps of this approach. Here, lines 1 to 4 are equal to the BASICPACKING algorithm. Then, in lines 5, the algorithm gathers all positions from the template. In lines 6 to 12, the algorithm selects the closest position with respect to the data centers of the circles. It therefore allows the largest circle to choose its position first. The other circles have to choose among the remaining positions. This placement of extended circles improves the clustering and reduces the error of the distances among the points and the center of the extended circles. The largest circle shall induce the least error. Finally, in line 13, the algorithm

Algorithm 6.3 Function dataRelatedPacking for circle packing

Input: Set l of circles $\{c_1^+, \ldots, c_{|l|}^+\}$ with attributes x, y, r, n, cls, x_d and y_d
Output: An updated set of circles $\{c_1^+, \ldots, c_{|l|}^+\}$

1: circles $\leftarrow$ SORT(l, *order by r desc*)
2: $r_{largest} \leftarrow$ circles[0].r
3: $(x_{center}, y_{center}) \leftarrow \left(\frac{\sum_{i=0}^{|l|} \text{circles[i]}.x \cdot \text{circles[i]}.n}{\sum_{i=0}^{|l|} \text{circles[i]}.n}, \frac{\sum_{i=0}^{|l|} \text{circles[i]}.y \cdot \text{circles[i]}.n}{\sum_{i=0}^{|l|} \text{circles[i]}.n} \right)$
4: template $\leftarrow$ createTemplate(x_{center}, y_{center}, $r_{largest}$, $|l|$)
5: positions $\leftarrow$ {getPosition(template, 1), ..., getPosition(template, $|l|$)}
6: **for** $c^+ \in l$ **do**
7: $\quad (x_{best}, y_{best}) \leftarrow (\inf, \inf)$
8: $\quad$ **for** $(x_d, y_d) \in$ positions **do**
9: $\quad\quad$ **if** $pdist\left((x_d, y_d), (c^+.x, c^+.y)\right) < pdist((x_d, y_d), (x_{best}, y_{best}))$ **then**
10: $\quad\quad\quad (x_{best}, y_{best}) \leftarrow (x_d, y_d)$
11: $\quad (c^+.x_d, c^+.y_d) \leftarrow (x_{best}, y_{best})$
12: $\quad$ positions $\leftarrow$ positions $\setminus (x_{best}, y_{best})$
13: **return** toSet(circles)

returns the updated set of extended circles.

Space-Optimized Packing Space-optimized packing tries to improve the output of the basic packing in order to generate smaller placeholder circles. The idea is to move the circles closer to the center of the placeholder circle to decrease its radius. By doing so, there is less dead space in a packed circle, and thus on the map. Consequently, space-optimized packing improves the quality of the overall clustering. Figure 6.5 exemplifies space-optimized packing. In contrast to the basic packing specification on the left side, the circles on the right side are moved to the center to form a condensed representation where the dead space is removed.

Algorithm 6.4 shows the steps of this approach. The lines 1 to 6 are equal to the basicPacking algorithm. In line 7, the algorithm defines a constant step size for moving the circles closer to the center. If the step size is too small, then there will be a lot of iterations. If the size is too big, then it is unlikely that the algorithm finds a circle that moves into the direction of the center. Our suggestion is to set the step size with respect to the screen pixels, e.g. to one pixel. In line 8, the algorithm introduces the variable *result* that serves as result set. In lines

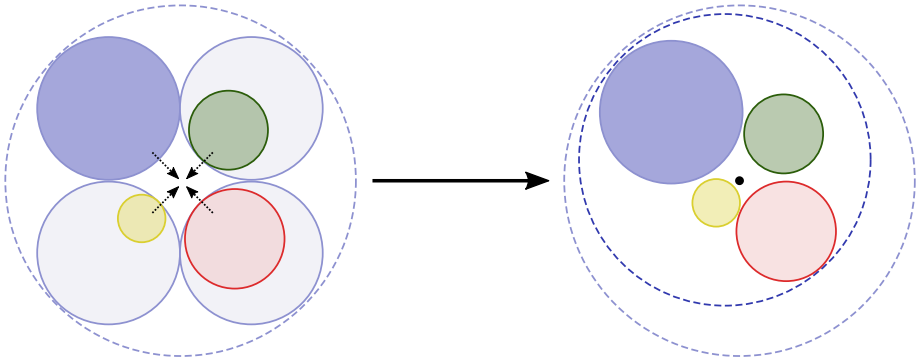


Figure 6.5.: This figure illustrates space-optimized packing. The circles on the right side are moved towards the center so that they are more compact and the dead space is reduced.

9 to 19, a loop moves the circles closer to the center in a round-based fashion, starting with the largest circle. In line 11, the movement towards the center is temporarily kept in (x_{new}, y_{new}). In the following, it should be remembered that *cpdist* calculates the distance between a circle and a point and *cdist* calculates the distance between two circles, including their radii. In lines 12 to 14, the algorithm checks whether the center of the placement template would then be in the extended circle. If so, we remain the old position of the extended circle and continue with the next extended circle. In a similar way, the algorithm checks for overlap with other extended circles, and if so, it continues with the while-loop. In lines 15 to 18, the algorithm checks whether the movement did not introduce an overlap with another circle. If the movement is valid, then the algorithm updates the coordinates of the circle in-place in line 19 and appends it back to the queue in line 20. In the other case, the result set is filled (lines 13 and 17) and eventually contains all circles. Finally, in line 21, the algorithm returns the resulting set of circles.

Data-related Space-Optimized Packing The data-related space-optimized packing method is a combination of the former two approaches. First of all, it finds a placement in the template for q circles based on the data-related approach. Then, in order to optimize the placeholder circle, it uses the space-optimized approach. Our hypothesis is that this approach works best as long as both individual approaches improve the quality of the output. However, it is the most costly algorithm.

Algorithm 6.4 Function spaceOptimizedPacking for circle packing

Input: Set l of circles $\{c_1^+, \ldots, c_{|l|}^+\}$ with attributes x, y, r, n, cls, x_d and y_d
Output: An updated set of circles $\{c_1^+, \ldots, c_{|l|}^+\}$

1: circles $\leftarrow$ SORT(l, *order by r desc*)
2: $r_{largest} \leftarrow$ circles[0].r
3: $(x_{center}, y_{center}) \leftarrow \left(\frac{\sum_{i=0}^{|l|} \text{circles[i]}.x \cdot \text{circles[i]}.n}{\sum_{i=0}^{|l|} \text{circles[i]}.n}, \frac{\sum_{i=0}^{|l|} \text{circles[i]}.y \cdot \text{circles[i]}.n}{\sum_{i=0}^{|l|} \text{circles[i]}.n} \right)$
4: template $\leftarrow$ createTemplate(x_{center}, y_{center}, $r_{largest}$, $|l|$)
5: **for** $i \in |l|$ **do**
6: (circles[i].x_d, circles[i].y_d) $\leftarrow$ getPosition(template, i)
7: step $\leftarrow$ *define a step size*
8: result $\leftarrow \emptyset$
9: **while** $\neg$EMPTY(circles) **do**
10: $c^+ \leftarrow$ POP(circles)
11: $(x_{new}, y_{new}) \leftarrow \left(\begin{pmatrix} c^+.x_d \\ c^+.y_d \end{pmatrix} + step \cdot \frac{(x_{center}, y_{center})^T - (c^+.x_d, c^+.y_d)^T}{|(x_{center}, y_{center})^T - (c^+.x_d, c^+.y_d)^T|} \right)^T$
12: **if** $cpdist\left((x_{new}, y_{new}, c^+.r), (x_{center}, y_{center})\right) < 0$ **then**
13: result $\leftarrow$ result $\cup$ c^+
14: continue whileLoop
15: **for** $c_{other}^+ \in$ (circles $\cup$ result) **do**
16: **if** $cdist\left((x_{new}, y_{new}, c^+.r), (c_{other}^+.x_d, c_{other}^+.y_d, c_{other}^+.r)\right) < 0$ **then**
17: result $\leftarrow$ result $\cup$ c^+
18: continue whileLoop
19: $(c^+.x_d, c^+.y_d) \leftarrow (x_{new}, y_{new})$
20: APPEND(circles, c^+)
21: **return** result

Experiments

This subsection presents different experiments with respect to the circle packing extensions of *CMQ*. First, we show and discuss the results of two runtime measurements. Then, we look at the change of visual point clustering quality with respect to single class data. Finally, we present the results of a user study that aims to validate the quality results.

We created 18 multi-class data sets by combining 10 data sets from GBIF (cf. Chapter 5) in different variations. Table 6.1 shows the data set names and their number of points. We chose from $c \in \{1, \ldots, 6\}$ classes and retrieved each time c

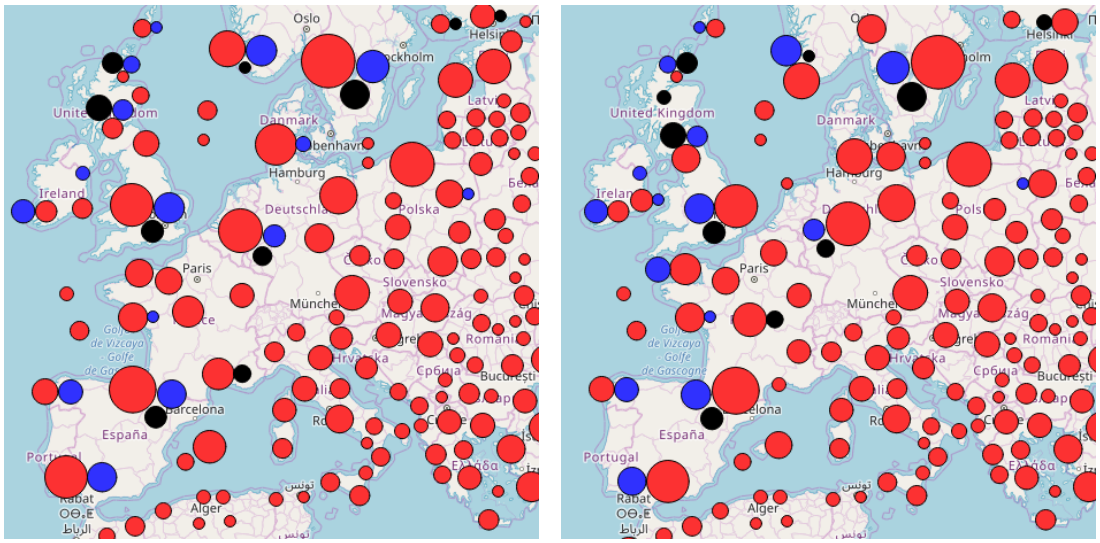


Figure 6.6.: This figure shows two screenshots of *CMQ* with circle packing. The left screen displays the result of basic packing and the right side the result of data-related packing.

data sets from the original set uniformly at random. Then, we concatenated the c data sets into one new data set for evaluation within which the points were assigned to class labels $1, \ldots, c$. Figure 6.6 shows one of those multi-class data sets. Here, we see three classes of data distributed in Europe. One can see the differences between basic packing on the left side and data-related packing on the right side. For instance, the different arrangement of the circles in Great Britain allows a more accurate representation of blue circles for the coverage of Ireland. This was possible because in England, where the large red circle moved east, a different arrangement appeared. For all experiments, we have used a system with an AMD Ryzen 7 2700X CPU running at 3.70 GHz and 32 GB of RAM.

Table 6.1.: This table shows the names of the data sets and their number of points for generating the evaluation data set.

Data Set Name	# Points	Data Set Name	# Points
Tayloria tenuis	403	Camaroptera brachyura	37 238
Carduus personata	1 994	Deschampsia flexuosa	188 985
Platyperigea montana	2 537	Acrocephalus scirpaceus	459 473
Stachys alpina	7 826	Spizella passerina	1 653 584
Carassius gibelio	16 841	Larus delawarensis	2 329 401

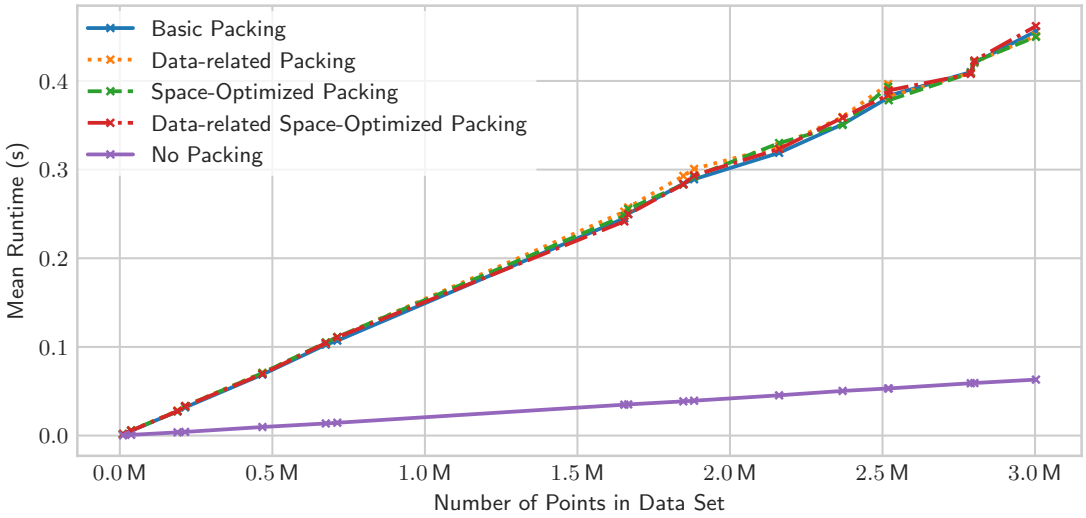


Figure 6.7.: This figure shows the runtime of *CMQ* with the different packing variants and without packing.

Runtime We conducted two runtime experiments. The first experiment tries to gather information about the scalability of the method with respect to the number of points in a data set. Here, we use the 18 data sets that contained different numbers of classes and average the runtime of zoom levels $z \in \{1, \ldots, 5\}$ and five repetitions each. Figure 6.7 shows the resulting runtimes. The x-axis reflects the number of data points in the data set. The y-axis reflects the average total runtimes of the four *CMQ* variants with packing and the variant without packing. We draw lines that interconnect the different entries in order to recognize trends.

The plot contains three different results. First, the runtime with respect to the number of points is linear for all methods. Second, the variant without packing is much faster since the runtime reflects a much lower constant increase per additional data point. Third, it is not apparent that any packing method outperforms another one. The differences between the lines are insignificantly small.

The second experiment shows the impact of the number of classes on the runtime of the *CMQ* algorithm. Here, we generated another 60 data sets that we extracted out of the same ten data sets as the prior ones. Instead of combining the data sets to gather multi-class data, we assigned class labels to each record uniformly at random. Here, we used numbers from one to six again. Figure 6.8 shows the results of the runtime measurements in the form of a bar chart. The x-axis denotes

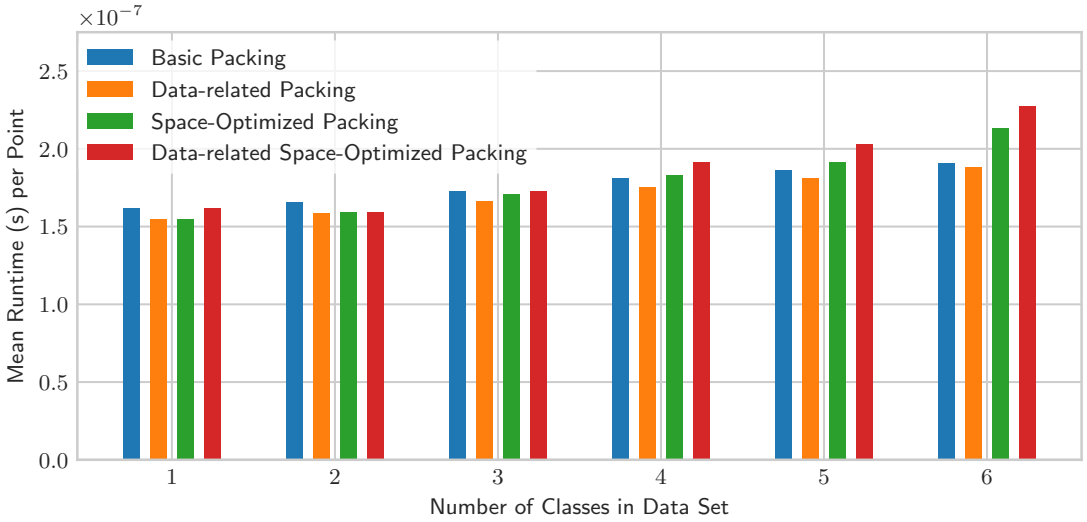


Figure 6.8.: This figure shows the runtime for the packing variants for different numbers of classes in the data sets.

the number of classes. The y-axis shows the mean runtime per data point over all data sets and zoom levels. We have calculated this number by dividing the total runtime by the number of data points in the individual data sets. For each of the four packing variants we display individual bars.

As a first finding, the bars show that the runtime per point increases for all packing variants. Only data-related space-optimized packing has an outlier for one class, although it has to choose a data-related position for only one circle. The second finding is that the basic packing seems to have a small overhead for a small number of classes, but has less increase for a larger number. This is because it does not have to do any effort for choosing the right template spots within the placeholder circle nor does it have to optimize the positions afterwards. For the other variants, space-optimized packing seems to have the largest increase alone and also in combination with space-optimization. A reason for this is that the incremental movement towards an optimal position within the template circle requires more time per class than choosing the best spot for one of the circles like in data-related packing.

Quality For analyzing the quality, we evaluate the 18 data sets from the introduction of this experiment section again and compute the quality values. Since most quality scores are similar, we only show the differences in Figure 6.9. Here, we see that *CMQ* without circle packing obviously produces overlaps within the

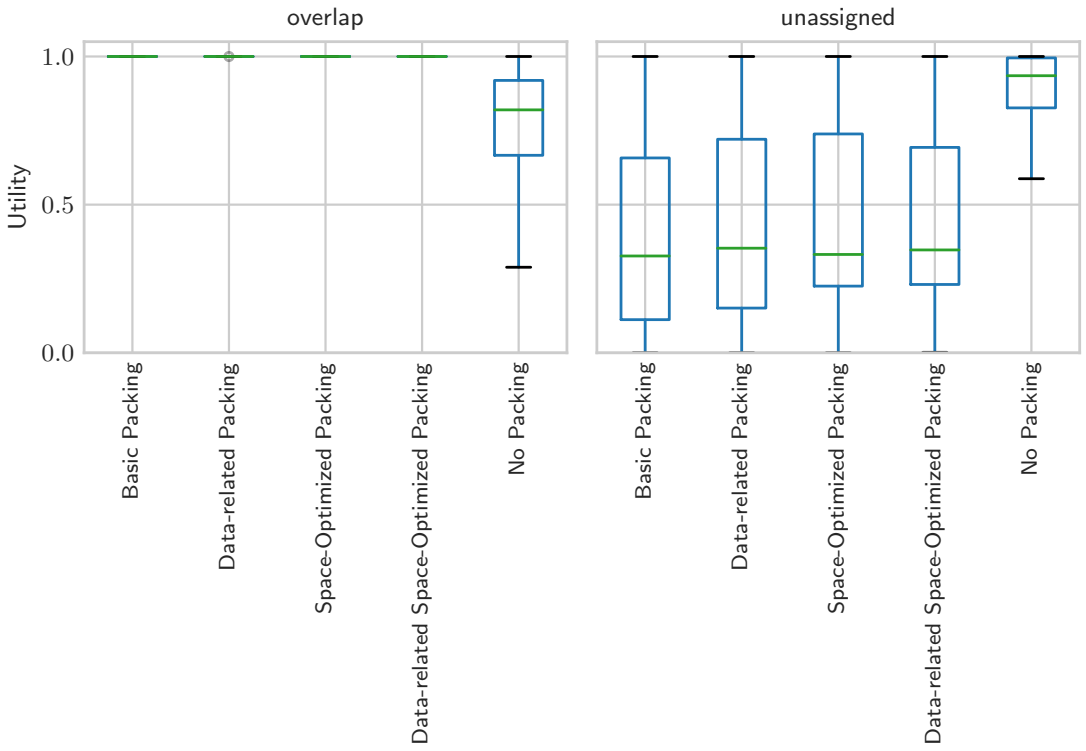


Figure 6.9.: This figure shows quality scores of the measures *overlap* and *unassigned* for the circle packing variants.

circles of different classes. In contrast, *CMQ* with circle packing has a lower score in unassigned points. An explanation for this is that a displacement of circles in the circle packing step produces a distance to the actual, underlying circles. Thus, the enclosing assignment, which is used for calculating this score, retrieves less assigned points for the individual circles. This seems to be an unavoidable trade-off when using circle packing to display multiple data sets. Without having overlaps, it is impossible to achieve a high score here. The scores among the circle packing variants have only few differences and those are hardly noticeable.

Figure 6.10 shows the differences in the average quality score of the circle packing variants of *CMQ*. Here, the variants all produce similar scores. However, the means show some subtle differences. Basic packing has the lowest mean with 0.617813. Then, data-related packing has a mean of 0.618316 and is followed by data-related space-optimized packing with 0.619437. Among the packing variants, space-optimized packing scores highest with a mean of 0.623014. Surprisingly, the variant without packing produces the highest mean score of 0.635479. By reconsid-

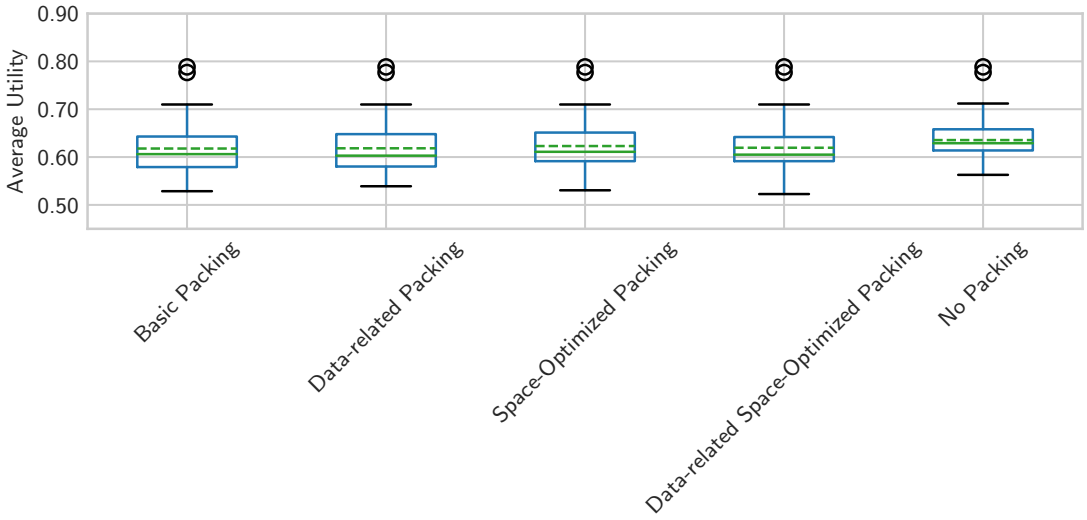


Figure 6.10.: This figure shows the average quality score for the circle packing variants.

ering the trade-off between the scores for unassigned points and overlap, we notice that the overlap problem is underweighted in this average score. To investigate this further, we will present a user survey in the following that incorporates actual user impressions with respect to this problem.

User Survey In order to get real-world feedback from users, we conducted a survey among 75 computer science, data science and business informatics students. The survey consisted of 20 questions, which were partitioned into two blocks. The first block tried to find out whether users prefer circle packing over no packing. The second block consisted of questions regarding preferences for basic packing and data-related packing. Due to time constraints in the inquiry, we decided not include questions regarding space-optimized packing. However, this setup allows us to investigate if the small increase in the computed quality score of the measures from basic to data-related packing is also reflected in the user survey.

Each time, we showed the participants two maps that were generated using different algorithm variants of the same multi-class data set. We chose the data sets out of the 18 data sets with different zoom levels again that were used in the previous subsection. The maps were accompanied by an explanatory textual description about the distribution of these data sets. Furthermore, there were supplementary maps of the single data sets. The participants had the option to decide whether

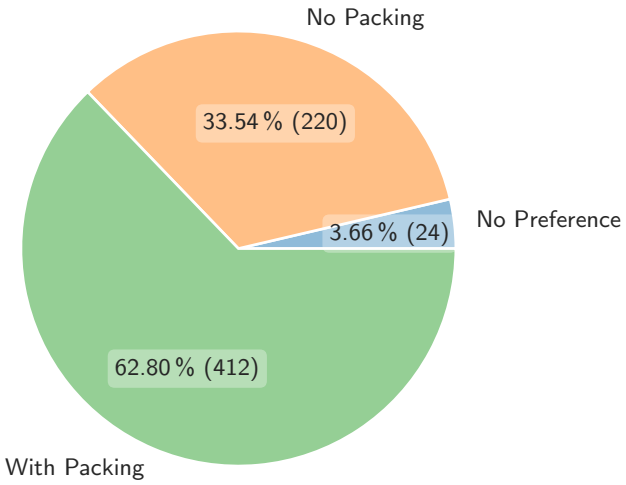


Figure 6.11.: This figure shows the survey results for maps that were created by *CMQ* with circle packing versus maps that were created without circle packing. The value next to the percentage shows the absolute number of votes.

they preferred the left or right map. In addition, they had the option to be indecisive and not to prefer any representation. This allowed us to gather direct A/B comparisons about the preferences.

All questions were shuffled uniformly at random and presented to the participants. Note that not every participant answered every question. The questions with the minimum number of answers had 63 responses.

Figure 6.11 shows the aggregated results in the form of a pie chart with respect to maps that were created with or without (basic) circle packing. We clearly see that our hypothesis that users prefer circle packing over the unpacked variant is confirmed. Roughly $^2/_3$ of the users prefer circle packing in one-to-one comparisons. On the other hand, also $^1/_3$ of the users prefer maps without circle packing and accept some overlap of the circles. This finding can be interrelated with the quality scores of the previous subsection that presented a trade-off between circle overlap and unassigned points.

Figure 6.12 presents the results with respect to the comparison of basic to data-related circle packing. Data-related circle packing clearly outperforms basic packing by receiving more than half of the votes. Basic packing is preferred in less than 30 % of the responses. In contrast to the previous comparison, the number

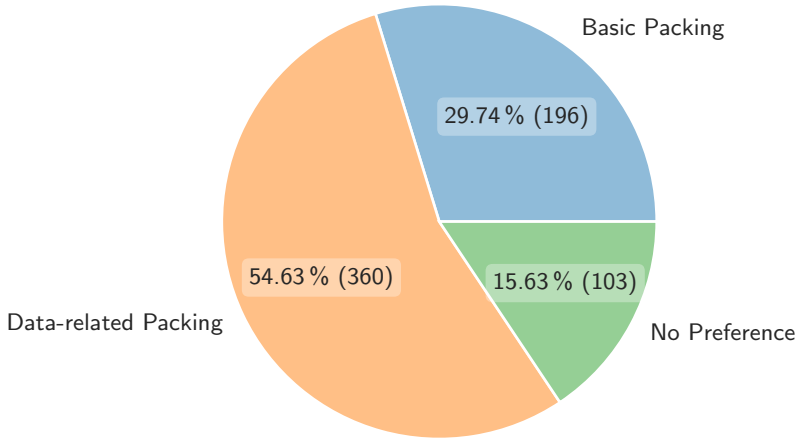


Figure 6.12.: This figure shows the survey results for maps that were created by *CMQ* with basic circle packing versus maps that were created with data-related circle packing. The value next to the percentage shows the absolute number of votes.

of votes with no preference is quite high with 15 %. This is in accordance with the close quality scores of the previous subsection.

The results show a clear tendency that circle packing is preferred over no packing and that it is worth investigating improvements of the packing method. Furthermore, they show that the results of the quality measures are connected to the answers from real users.

Quality Criteria and User Survey Since we have our quality criteria on the one hand and the binary results of the user survey on the other hand, it is complicated to get a common interpretation. We therefore utilize techniques from preference learning [Hül+08], which aim to find a predictive preference model from observed preference information. We employ this method to bridge the gap between criteria and survey results, and to gain additional insights. Fober et al. [FCH11] propose a method that uses pairwise labels and applies multi-objective optimization to find a set of utility functions. These functions model latent user preferences and are in accordance with the provided data. The proposed method transfers the problem to a binary classification problem and uses gradient descent to gradually improve upon positive and negative examples. In our case, we insert the labels from the (in

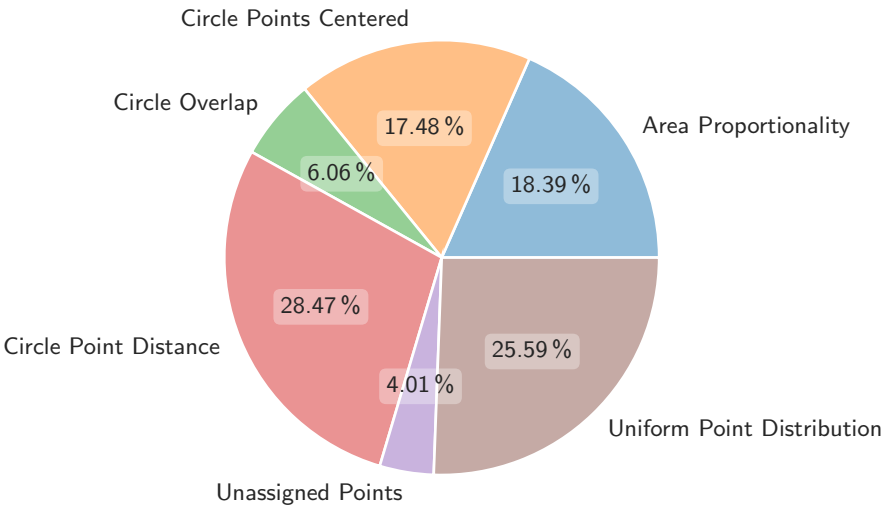


Figure 6.13.: This figure shows the influence of the quality criteria as a result of preference learning based on the user survey result.

total) 1 187 answers to our 20 questions in the form of $(winner, loser)$ pairs. Additionally, we insert the utility values provided by our multi-class quality measures. We calculate these by computing the quality for the underlying data sets for each of the two options for each question. The preference learning algorithm then finds a set of weights and a joint linear utility function based on the preferences of the user survey. We can normalize the weights and use them as an indicator of influence for each quality criterion on the user decisions.

Figure 6.13 shows the resulting influences in form of a pie chart. First of all, there is no quality criterion that has no influence on the user preferences. This means that all of them are valid for assessing the algorithmic results. Furthermore, *circle point distance* has the largest influence, followed by *uniform point distribution*. These criteria evaluate if the circles are close to the underlying data and whether they are skewed. Then, *area proportionality* and *circle points centered* seem important factors for the user preferences. In particular, the former criterion allows differentiating the hot spots from the sparse regions in the data. The latter criterion ensures that the circles are located at the correct region on the map. Finally, *circle overlap* and *unassigned points* seem to have the least influence.

Regarding overlap, the survey results show the tendency to allow some degree of overlap as long as it contributes to the overall understanding of the map rep-

resentation. This means that it could potentially make sense in the multi-class case to introduce a relaxation of this criterion. As an example, the query procedure could incorporate another ϵ_{relax} parameter that specifies a small error tolerance. Thus, *CMQ* would only merge overlaps that lead to severe occlusion on the map.

While the influence values that were derived from preference learning are useful indicators for the influence of the quality criteria, there is most likely some bias in the survey data. For instance, all packing variants produce *circle overlap* values of 1, whereas only the non-packed representation produces lower values. This means that the parameter is uncorrelated to the preference for all questions regarding different packing variants. For *unassigned points*, the question design did not especially map the raw data in a different visualization such that survey participants had the chance to see unassigned points other from the textual description. This means that the answer based on their preference was most likely not affected by this criterion.

In conclusion, preference learning is a suitable method to gain further insights into the assessment of the multi-class quality criteria based on preferences from real users. Relevant findings were that all criteria have an influence on the preference and that the influences are different among the criteria. However, an extended and carefully balanced survey design could increase the significance of the findings.

6.3. Summary

This chapter presented extensions of the *CMQ* algorithm. First, it showed means to cope with miscellaneous attributes that were of non-spatial nature. For this, the chapter contained descriptions of different methods for numerical as well as textual attributes.

Second, the chapter introduced *CMQ* extensions for visualizing multi-class data. Here, the section contained method descriptions for pie chart maps and circle packed maps. For circle packing in particular there were different improvements introduced with respect to a method from literature [Jän+12]. For all circle packing variants, we conducted runtime and quality evaluations. Especially regarding quality, we conducted a user survey that gave an extended insight into the perception of the *CMQ* variants. Additionally, we evaluated the survey results using preference learning techniques that yielded influences of quality criteria with respect to the answers of the survey participants.

7 The VAT System

After presenting a particular visualization method in the previous chapters, this chapter presents the design and the development of the **V**isualization, **A**nalysis and **T**ransformation (Vat) System. This embeds *CMQ* in the context of a system that relies heavily on these and similar methods. It starts with a motivation in Section 7.1. Here, it discusses key aspects of a modern geographic information system and the necessity to develop such a new system. Section 7.2 presents the main architecture of the Vat System. It highlights important design principles as well as concepts and gives an introduction to the components of the system. The main part is provided in Section 7.3 and contains detailed descriptions of important features and realizations of concepts of the Vat System. The primary focus is on the visualization component and the user interaction with the system. Section 7.4 presents the connection of the Vat System to the GFBio infrastructure. Besides providing details about the portal integration, we give a brief outlook on how to potentially use the Vat System in other research projects. In Section 7.5, we present an exemplary use case within the Vat System. Here, we follow the working path of a single user to achieve a research goal. Section 7.6 outlines the development process of the user interface. It presents two user studies that have been conducted on the path to the final design. In Section 7.7, we investigate related systems in the geo processing ecosystem. We divide the systems into a number of groups and highlight differences in comparison to the Vat System. Finally, we summarize this chapter in Section 7.8.

7.1. Motivation

In recent years, data-driven research has become increasingly popular [AAG03; Ste+13]. A major reason for this is the exponentially growing amount of data in all fields of science. This makes it necessary to investigate this data in order to find interesting correlations or anomalies. In the field of biodiversity and geography, a lot of data is still unused [CW14; Gri15]. The current landscape of geo processing

systems makes it difficult to incorporate existing geo data into research activities. After the completion of a research project, many research data sets run into the archives without being in compliance with standard data formats and can hardly be integrated without a great deal of manual transformation effort. However, the goal is to have a system that helps to integrate such data and to use the wholeness of research results for new analyses.

We have developed the VAT (**V**isualization, **A**nalysis and **T**ransformation) system to tackle this problem. It allows investigating different kinds of geo data in an exploratory fashion. To achieve this, it provides a map with a coordinate system to visualize the data in the first place. By following a visual analytics approach, VAT enables researchers to work with geo data, manipulate it and combine it by simultaneously providing visual feedback. The processing of geo data is quite complicated. In addition to different data formats and standards, the geographical representation and in particular the processing of multiple data sets with different projections holds several pitfalls.

VAT aims to support different types of users. For instance, a computer scientist has the ability to develop fast data processing, but lacks in experience in working with geographic data. A geographer, for example, has the necessary expert knowledge in working with geo data, but has difficulties in efficiently processing large data sets. As a third example, a biodiversity researcher is a specialist in producing a meaningful analysis of the distribution of species, but needs support in working efficiently and correctly with geo data.

Since interactivity is a crucial component of geo analysis tools [Rot13], VAT provides means for interactively investigating data. Following Shneiderman's information seeking mantra [Shn96] (cf. Section 3.2), it provides an overview of the data, tools to zoom and filter as well as the option to retrieve more details on demand. This includes data aggregation techniques such as the Circle Merging Quadtree (*CMQ*) for point data as presented in the prior chapters. Furthermore, it provides simplifications of data to achieve quick response times for results in preview quality since this quality is sufficient for visualization. To complete the interactivity methodology, VAT incorporates fast, GPU-based algorithms to leverage modern hardware infrastructures.

An essential feature of VAT are tools for combining data of different types, such as raster and vector data (cf. Section 3.1). This allows covering the heterogeneity of geo data, and thus making sense of data and gaining insight from data from various sources. Another important feature is the inherent temporal support, which allows users to process time series of data. This is neglected by most contemporary geo information systems (GIS).

Vat provides a modern, web-based graphical user interface (GUI) that uses state-of-the-art web components. This leads to an intuitive user interface that does not require any local installation for the user. By means of this connection to the backend, users can on the one hand upload their own research data (pre-publication data) and process it in the system. On the other hand, they can incorporate existing research data and geographical data products that are provided by Vat, e.g. essential climate variables derived from ESA satellite images [Hol+13]. The provision of data drastically reduces the effort for a combined analysis, since it represents one of the most expensive parts of the processing chain [Lon+05].

Within the project scope of GFBio (cf. Section 1.1) Vat provides a close integration to the GFBio data centers that facilitates to reuse all research data that is made available by the participating organizations. Furthermore, since GFBio offers a joint venture of research collections and archives, the amount of mobilized data is steadily increasing. Users can search for and look up GFBio data in a warehouse-like fashion [BS97], and import it into Vat. For all computations, Vat offers a complete data lineage and provenance tracking. This makes it possible to generate all necessary citations from the sources, which is a decisive feature in science [EGA18].

7.2. Vat – Architecture and Data Model

The Vat System is a geo processing system that consists of two main parts. First, it consists of a high-performance back end called Mapping (**M**arburg's **A**nalysis, **P**rocessing, and **P**rovenance of **I**nformation for **N**etworked **G**eographics) and, secondly, of an interactive web-based user interface called Wave (**W**orkflow, **A**nalysis and **V**isualization **E**ditor). Figure 7.1 depicts the overview of the general architecture. Users, in general, work with Wave to interactively explore and process geographical data. Wave then interacts with Mapping which itself accesses the underlying repositories and databases that hold the source data sets. In addition to the user interface access, expert users and machines can directly communicate with Mapping via its API [Aut+15a]. The API consists of OGC standard protocols [Ope07] for accessing the data and a custom API for creating workflows and triggering their processing as well as miscellaneous functionality like user and project management. Mapping implements OGC protocols like WFS [Ope10b] for querying vector data like points and polygons, WMS [Ope06] to create map images out of raster data and WCS [Ope08] to output raster data in a machine-readable

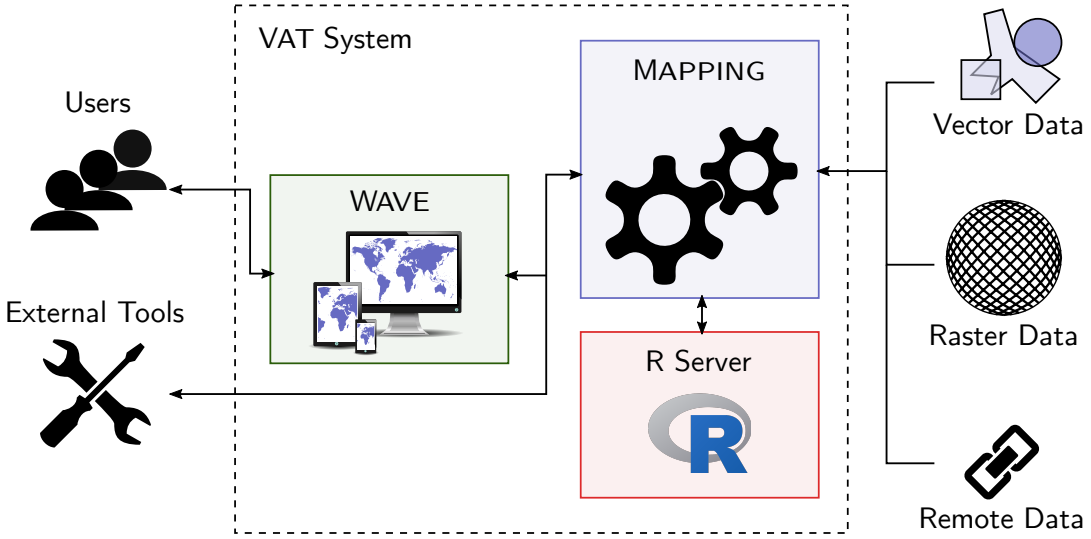


Figure 7.1.: This illustration presents a concise view on the system architecture of the VAT System.

format. WAVE hides all complexity of the underlying system and lets users explore the data and incrementally build processing workflows.

The VAT System supports two very different data types: raster and vector data (cf. Section 3.1). A raster is a uniform grid of numeric values. Cells contain either continuous values, e.g. temperature measurements, or discrete values, e.g. a classification based on land usage. Vector data is modelled to be consistent with the Simple Feature Access model [Ope10a]. Here, a data set consists of a set of features with attributes. Since we want to achieve clean semantics for our operators, we only allow homogeneous collections of features. Thus, a data set consists either of points, lines or polygons. Each feature is optionally a multi-feature consisting of multiple points, lines or polygons. A multi-feature has then a shared set of attributes. We implemented the attributes for each feature as a key-value map. VAT supports textual and numeric values. For raster values and numerical feature attributes, we introduced the notion of a *unit*. Units specify what was measured (e.g. temperature in °C) and have metadata if the values correspond to a classification or to a continuous variable.

In the VAT System all data always has a temporal context in addition to the spatial context [Bei+17c]. The joint context describes the validity of a data item in terms of a *location* and a *time interval*. In the same manner, a query specifies a spatial bounding box and a temporal interval. Figure 7.2 illustrates what we call *spatio-temporal query rectangle*. Here, we query Europe in the years 1990 to 2000. The system returns all results that are valid within these constraints. As time is

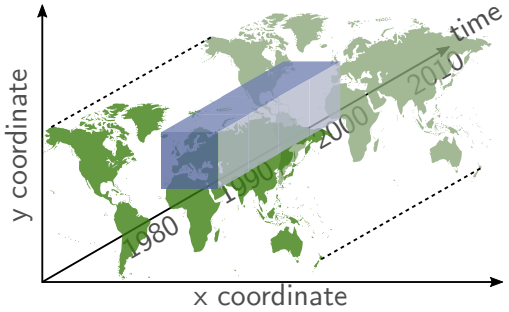


Figure 7.2.: An exemplary spatio-temporal query rectangle that spans over Europe in the time range of the years 1990 to 2000.

an integral part of our data model, we consider all data sets as time series. This implies that all operators in the VAT System must adhere to time series semantics, e.g. Allen's interval semantics [All83].

With respect to our data model, the temporal semantics have different implications. For our raster data type, we define that all cells of a raster have the same temporal validity. A raster data set then consists of multiple rasters with disjoint temporal validity. For the vector data types, each feature can have a different temporal extent. Within one feature all data has the same temporal validity. There is no restriction for a vector data set to have features of disjoint temporal validity. It would be impractical, for instance, if a data set could not contain multiple animal trajectories of January 2019.

The VAT System processes all data as workflows. A workflow consists of all inputs and processing steps and describes how they are interconnected. A query to the VAT System thus consists of a spatio-temporal query rectangle and a workflow.

The idea of the VAT System is that the user processes the data by starting with the sources and incrementally combining, filtering and transforming it until he reaches a result. A crucial component of the VAT System is that it builds the necessary workflow implicitly in the background such that the user does not have to take care of it. This helps in the exploration phase where users discard multiple processing and analysis ideas because they do not lead to the desired result. We call this pattern *exploratory workflows* (cf. Figure 7.3) [Aut+15b]. Since the exploration leads to incrementally built workflows, the VAT System leverages previous results by storing intermediate results in a caching data structure [Kör16]. This avoids many redundant and expensive recomputations. The retrieval of cached results also applies to changes in the spatio-temporal query rectangle. Subsequent queries that match, contain or overlap cached results can thus be answered faster.

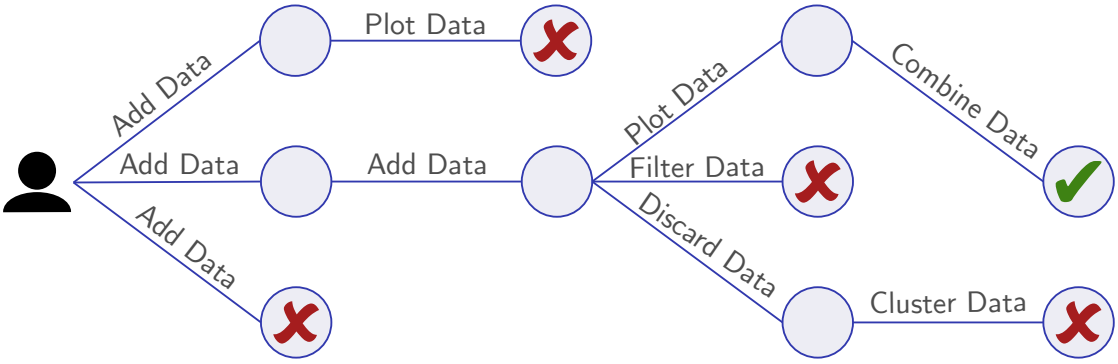


Figure 7.3.: The term *exploratory workflows* describes the transparent tracking of computational steps that have led to a result while discarding dead ends.

For example, when plotting variables of a data set that was created by enriching animal occurrences with attributes of a climate model, the enriched data set may already be cached. Furthermore, when querying first Europe and then Germany, the results can be extracted from the cache since the first result contained the second result.

The two main components of the VAT System are built upon various technologies. WAVE is a web project written in TypeScript [HL16]. It uses Angular [SC17] as main framework that allows to write reusable web components and to structure the application according to the model-view-controller (MVC) [Gam+94] paradigm. For displaying the geo data on a map, we utilize the flexible OpenLayers [GSH15] framework that uses WebGL [Mar11] for efficiently visualizing different geo data types. It also works well with different geographical projections. In the next section we will discuss, among other things, several dynamic components, which we realized by using Rx/JS [DA17]. The Rx/JS framework allows us to build a reactive application by using powerful iterators and asynchronous events. For plots and graphs, we use the D3.js [BOH11] library that provides building blocks to create custom data visualizations.

The second main component of the VAT System, MAPPING, is written in modern C++. It uses a custom architecture which is enhanced by several libraries for networking (Boost[1], POCO[2]) and data import (e.g. the Geospatial Data Abstraction Library (GDAL) [War08]). A FastCGI [Bro96] interface provides a connection to a web server like the Apache HTTP Server [Bro96]. In order to exploit modern hardware, we implement the operators to execute highly in parallel [Aut+15b].

[1] www.boost.org
[2] pocoproject.org

To support multiple GPU vendors as well as many-core CPUs, we use OpenCL [SGS10] and its C-based programming language. Since the goal of the Vat System is to be used in multiple research projects, we have implemented a modular design. This design allows us to combine and compile distinct features for specific projects. Furthermore, an adapted user interface exists alongside adapted back end functionality.

7.3. Wave – Overview and Features

This section contains an overview of Wave, the user interface of the Vat System. This focus allows a user- and functionality-centric view on the overall system. We describe features that affect both the front and back end of the Vat System, and we will also address the interconnection to Mapping and Mapping's mode of operation along the way. This section starts with an overview and then discusses individual aspects in more detail.

7.3.1. Wave Overview

Wave is the front end of the Vat System. It offers an intuitive user interface to support interactive, exploratory geo analysis primarily for biodiversity data. The basic usage is as follows: Users upload either their own data or choose data from the data repository. The current version contains species occurrences points, environmental data like climate and elevation models as raster data as well as polygonal expert range maps or country borders. Then, they apply several operators to the data. This includes filtering the data, e.g. restricting species points to a country, and enriching data by combining, for instance, point and raster data. Additionally, users can transform data, which includes either geographical projections or data formats. For example, users can create a heat map raster from a set of points. Furthermore, users can use tools to create plots and diagrams from the data. Finally, users can share their data with other users or export it into their local workspace. At each step, users can review the workflow of processing as well as the aggregated citations of each intermediate result.

The central part of Wave is the map. The idea is to always visualize the geographical context of the data. We follow this approach because it is in accordance with the concept of visual analytics (cf. Section 3.2) and can exploit the spatial nature of the data. The map can display the data in different map projections, since different projections make sense depending on the nature or location of the

data set (cf. Section 3.1). In addition, users can utilize a background layer, e.g. an OpenStreetMap backed country border map or a topographic terrain map. This is common to most GIS systems and expected by users since it facilitates map orientation. Of course, users can pan the map to shift focus and zoom into interesting parts of the map or zoom out to get an overview. All data on the map can have an individual coloring, e.g. a color palette for a climate raster or a point color to distinguish multiple point data sets. For raster data sets, the coloring is calculated in Mapping and the resulting image is displayed on the map. Vector data is colored locally on the web browser.

The elements on the map are displayed in different layers. One layer corresponds to one data set or displayable entity on the map. This entity is the result of a workflow, which we described briefly in Section 7.2. For each layer we display its name and its legend in a list on the left side of the screen. The legend shows if the layer displays icons, classes or gradients and provides, for instance, a mapping from class color to class name. Users can reorder the layer list, which reflects in a reordering of the elements on the map. This is especially advantageous if one layer hides information of another one, e.g. if users want to display a set of points on top of a climate raster. A context menu offers options among others to hide or remove layers, or to change the appearance (symbology) of a layer. For instance, a user can color a point layer consisting of a data set of species occurrences such that each subspecies is colored differently.

Since it is not possible to display all non-spatial attributes of a layer on the map, the bottom of the screen contains a panel with tabs for a data table and a citation view. A user can select a single layer by clicking onto its list entry in order to retrieve the miscellaneous information. We call this concept *active layer*. The data table entries and the map elements are interconnected. Hence, a user can select a table entry and highlight it on the map and vice versa.

The right side of the screen contains a menu panel, which is optionally collapsible to increase screen space. It contains menu items for authentication, data insertion or upload, geo operators and project settings. Furthermore, it contains another panel for non-mappable items, which are data plots.

The authentication form in the menu panel links to the user database of the Vat System. This allows foremost that all progress is automatically stored on the server. Hence, users can continue their work from any system as long as they are logged in. Without login, the progress is stored locally on the user's web browser. Furthermore, a login allows users to manage their work in different projects. Each project consists of local settings like a list of layer and plots, a temporal reference, a map projection and a name. Users can then switch between

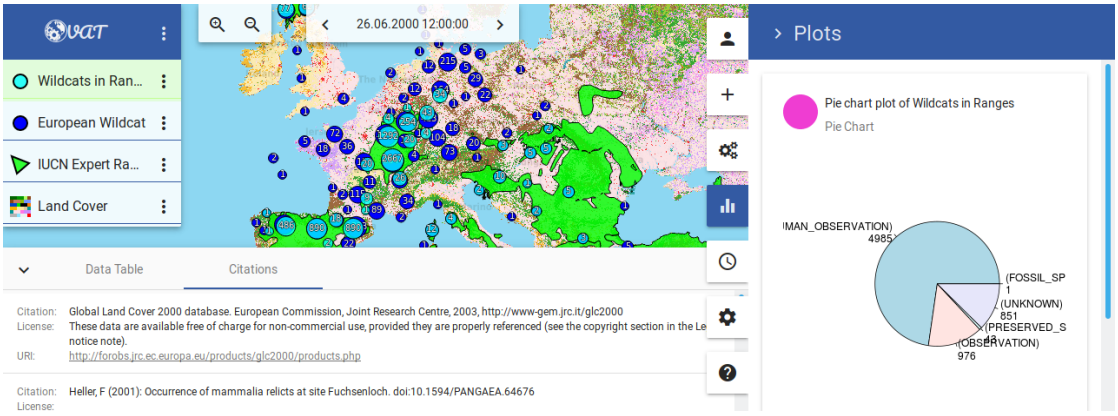


Figure 7.4.: This screenshot provides an overview of the main parts of Wave. There is a layer list on the left side, a list of citations at the bottom and a plot panel on the right side. Additionally, there is a zooming and temporal reference toolbar on top.

different projects and the system automatically adapts their workspace accordingly.

The top of the map contains two action panels. The first one provides buttons for zooming in and out of the map. This is, of course, also possible via scrolling or touch gestures on mobile devices. The second one is a reference time toolbar. Users are able to specify a reference point in time or a time range. The data on the map is computed accordingly to this specification.

Figure 7.4 shows a project that contains four layers and one plot. The layers are of different types and contain point and polygon collections and a raster time series. The global reference time is set to June 2000. The bottom tab panel displays a list of citations and contains a switch to display a data table instead. On the right, there is currently the list of plots visible. The boxplot on the right side shows classified data with respect to elevation data from one of Vat's data repositories.

7.3.2. Operators and Workflows

The core functionality of the Vat System is to manipulate and analyze geo data. Users do this by applying operators to the data. Here, each layer serves as possible input and the active layer is pre-selected if it represents a valid input for the operator. Each operator specifies which layer types are required and allowed as input. For example, a point in polygon filter operator requires exactly one point

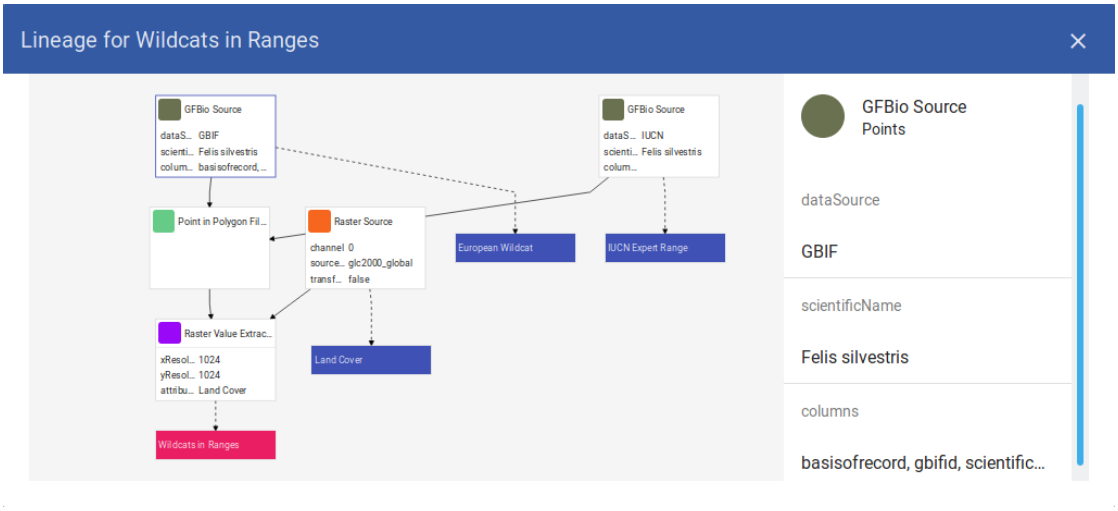


Figure 7.5.: This screenshot shows the representation of the workflow graph in Wave. Users can click on any operator to get more parametrization details.

layer and one polygon layer. This avoids specifying operators with false parameters. Other operators, e.g. the raster expression operator, require at least one raster layer. This means, users can select, for instance, two layers for calculating the difference between two rasters like the maximum and minimum precipitation for a year. The output of an operator is either a new layer or a plot. Hence, a filter operator does not necessarily replace the input layer. Instead, the input remains for inspecting the effect. Plots are unidirectional data elements in Vat, i.e., they are not usable as input.

During geo computations, it is necessary to match data, e.g. to match projections. Vat automatically introduces projections whenever necessary without manual engagement of the user. Since all projections of spatial data induce an error, Vat introduces projections as late as possible in the processing. Thus, data sets remain and are processed in their natural representation until it is no longer possible.

As described briefly in Section 7.2, Vat automatically records all operations like applying operators, plotting data and removing layers. Thus, at each point in time, we have the complete workflow of operations available. Figure 7.5 shows the workflow graph that we can query for each layer in Wave. It shows a tree from data sources over operators to a result. For each operator we can additionally review its parametrization. In future versions of Wave we intend to allow changes in workflow operators and make it possible to reintroduce intermediate results as a new layer.

7.3.3. Data Generalization and Exploration

There are two major objectives when interactively visualizing data for the user:

1. VAT aims to compute results in a near-real-time fashion for the user's exploration experience.
2. VAT provides an abstraction of the data that facilitates detecting interesting patterns.

Data generalization can address both concerns.

The generalization of raster data is possible by aggregating multiple adjacent cells and representing them in a lower resolution. This requires less space, but comes at the expense of losing information. However, the number of visible cells is naturally limited by the amount of pixels on the user's screen. Thus, it is sufficient to output raster images in this resolution for previews. Moreover, it is also sufficient to use source rasters and intermediate results of queries in this resolution instead of restricting the aggregation only to the results. This allows us to compute preview results with low latency. Users can afterwards trigger the computation in full resolution to produce scientifically valid results. In addition, users can incrementally improve the accuracy of the data processing and the data visualization by simply zooming into interesting areas. Zooming in increases the number of pixels in the map extent and, hence, the accuracy.

A popular approach to generalizing vector data is rasterization. This allows reducing the data to a fixed grid and makes a lot of operations much easier to compute, e.g. checking if a point is contained in a rasterized polygon. One drawback is the loss of attribute information that was attached to each feature. Especially varying textual data is hardly representable in a raster format. Another drawback is the loss of precision, e.g. by introducing a line width. A different approach is to apply simplification techniques that lead to fewer coordinates while still maintaining the general structure of the data. Apart from being expensive, this approach also causes semantic changes of queries. For lines and polygons it is possible use standard techniques like Douglas-Peucker [SGS10], but we aim for extending them to be able to consider topographic constraints in future work.

VAT offers an approach for generalizing big point data sets in order to speed up their visualization and to identify cluster patterns. As discussed in the course of the introduction in Chapter 1, displaying each point with its associated attributes exceeds the capabilities of current browsers on modern hardware even for sizes of less than one million points. Additionally, the size of transferring the data in the GeoJSON standard format stresses the internet connection of mobile devices. An

example for this are 23 039 kangaroo (Macropus giganteus) occurrence points from GBIF (cf. Section 1.2). The uncompressed size with 20 common attributes is $\tilde{1}5$ MB even for this relatively small data set. We can use the Circle Merging Quadtree (*CMQ*) here, which we presented in much detail in Chapter 5. It allows us to aggregate data to see hot spot clusters as well as outlier data. *CMQ* allows combining nearby data points dependent on the zoom level and map resolution. By zooming in the user gets a smaller excerpt of the map in more detail such that clusters break up and more details reveal. We represent the clusters as non-overlapping circles with logarithmically scaled areas based on the number of included points. VAT uses currently the Log_{10} transformation function as described in Section 4.5. In addition to the varying circle sizes that depend on the associated number of points, the circles contain the number of points as labels.

7.3.4. Linked Visualization and Data Table

WAVE is a reactive web application. With the map-centric approach it allows us to align the other data views accordingly. This means that whenever the reference time changes, all visible plots as well as the data table entries recompute automatically. This allows stepping through time in, for example, yearly intervals and keeping everything else synchronized. Classical GIS would need the user to compute all other items individually. In addition, all data views correspond to the current viewport, which is the extent of the map. Thus, a global view leads to a global computation of the data and, for instance, to zoom in on Germany, leads to recomputations for this selected area only. This facilitates the intuitive investigation of interesting places and the access to more details for certain areas on demand.

In accordance with most plots, the data table has to display aggregated data, e.g. clustered point data. Each table row corresponds to the visible, aggregated data points on the map. This allows us to limit the transferred data from MAPPING to the aggregated values only. Therefore, the table shows exactly the same data points as the map in the same resolution. Because of the geographic aggregation of the points, we also need to aggregate all other attributes of the data set.

In Section 6.1, we presented different efficient methods for aggregating the miscellaneous attribute values. For numeric attributes, we use the mean and standard deviation. By zooming in, the number of points in a cluster decreases and so does the standard deviation. This means the information gets more exact by diving into the data. VAT keeps for textual attributes a small number of representative points (typically three to five) for each cluster. Among the many options for selecting

representative points, we currently use the attributes of the points closest to the cluster center. The reason for our choice is that the accuracy of this information improves when zooming in.

Some textual attributes contain links to multimedia data. An example is a hyperlink to a photo of a data set object. Wave is able to detect multimedia links in a table cell and provides image views as well as audio and video players whenever relevant to access the data at its source. Thus, users are able to investigate the different aspects of a data set directly within the Vat System. We display all multimedia data in the menu panel on the right side of the user interface.

7.3.5. Citations and Provenance

Citations are very important for scientific work. They allow researchers to classify, comprehend and reproduce published results. They are also important for the publishers of those results, as they facilitate assessing the impact of their work. Aside from scientific results, also raw data has to be properly cited for the above reasons. Today, more and more organizations and journals encourage publishing data as a citable publication to facilitate data sharing. Articles, like Buneman et al.'s [BDF16], stress the importance of citations in scientific data management. Instead of applying complementary techniques to existing systems, Vat treats citations as first class citizens.

Our system aggregates all citations of the involved data sets. It exploits the automatically tracked workflows (cf. Section 7.3.2) to retrieve information from all incorporated data sources. We call the combination of

1. citation
2. license
3. URI (e.g. a link to a landing page)

provenance information. All operators are responsible for collecting or forwarding the provenance information of the result. For source operators this means that they have to harvest their sources for citations. On the other hand, processing operators combine the provenance information of their inputs via a duplicate eliminating union operation. This behavior can be altered for specific operators, e.g. a filter operator does not necessarily need to output citations of filtered out entries.

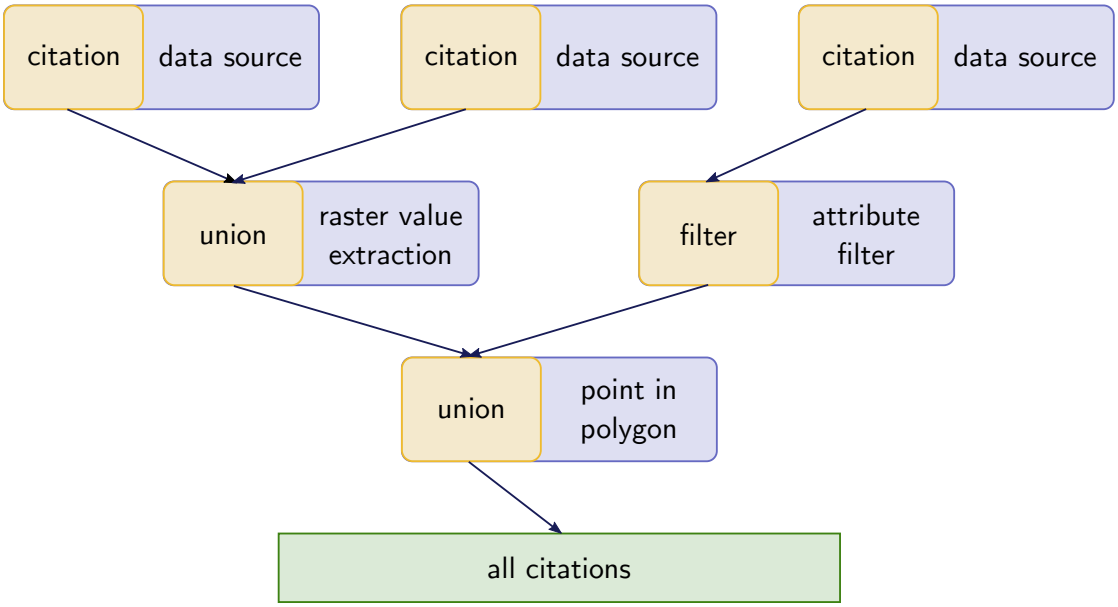


Figure 7.6.: This tree illustration visualizes the secondary computation path for citations that are computed alongside the actual data.

In general, the citation generation can be seen as a second workflow that calculates the citations alongside with the actual data. Figure 7.6 shows this extended workflow view. In Wave, the provenance information for the active layer is available in the citation tab at the bottom. In addition, Vat will always add this information to a data export.

7.3.6. Temporal Operations and Aggregation

Dynamic time series processing based on workflows is a core feature of Vat in contrast to other GIS-like systems. Our system supports the specification of date and time as a global temporal reference. As stated in Section 7.2, each query consists of a workflow and a spatio-temporal query rectangle. The time interval, in particular, expresses the validity of the result. The temporal reference slices a time series result such that it only contains elements that are valid at the given point in time or time interval. This facilitates an exploratory, on-demand processing of workflows in which Vat only generates the results for actually requested data. Furthermore, workflows can be reused for queries with different spatio-temporal query rectangles.

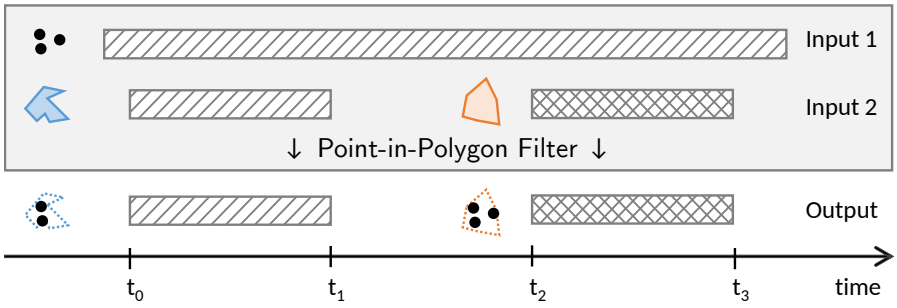


Figure 7.7.: This figure depicts the time series computation of a point-in-polygon filter [Bei+17a]. Users can query with different time slices that determine the results.

For example, a user may add the WorldClim[3] mean annual temperature data set as a raster layer to a project. This data set is a time series that contains monthly climate variables. By means of the temporal reference, the system is now able to choose the valid raster for the month of the currently selected point in time and add it to the map. Consequently, operations that include this data set also incorporate the correct raster with respect to the temporal reference. In comparison, traditional GIS oblige a user to manually add the correct raster from the data set beforehand. This is obviously a cumbersome and error-prone task. Here, a change of the temporal reference in the user interface of Wave automatically leads to the correct raster being displayed and computed.

The temporal validity of a data object is defined as an interval from a start time to an end time during which the object is incorporated into computations. The start and end times refer to a user-defined temporal reference system, which is in practice always the Gregorian calendar. When users want to combine data with different temporal validities, Vat may have to split data objects into multiple items with different validities. Figure 7.7 presents such a computation with an example of a point-in-polygon filter. Here, the point data set has an infinite temporal validity (always valid) and the polygon data set contains two features with different temporal validities. The resulting time series contains all points that are inside the blue polygon for the interval $[t_0, t_1)$. For $[t_1, t_2)$, there is no polygon and, hence, there are no points that are inside a polygon. For $[t_2, t_3)$, Vat outputs all points that are inside the orange polygon. When specifying a point in time in Wave, the query to Vat only returns the points that are valid within that time slice. Vat does not compute the whole time series and, thus, the diagram in Figure 7.7 displays a more theoretic view on the processing.

[3] www.worldclim.org

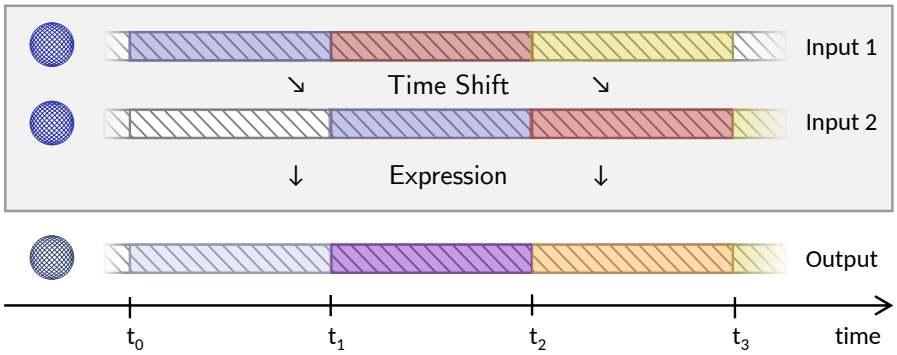


Figure 7.8.: This figure depicts a computation that first duplicates the input with a time shift operator (+1 time step). Then, it calculates a raster expression, e.g. an addition. The coloring emphasizes this process.

A different problem setting occurs when combining multiple data sets with non-overlapping temporal validity. Figure 7.8 exemplifies this by showing the computation of the temporal differences of the world population. Users can use the GPW[4] data set, which is a raster that represents population counts with a five-year validity. The idea is to compare the population with the population from ten years before. Users can apply a *time shift* operator in Wave that allows absolute or relative shifting of the temporal information of the spatio-temporal query rectangle. Thus, users add the GPW raster to the map and create a second layer by applying a relative time shift of -10 years to it. Then, an expression operator can calculate the differences between these two rasters. As a result, users can query a workflow with the year 2010 that computes the differences in population between 2000 and 2010. When stepping five years backwards in the temporal context in Wave, the system automatically computes the same workflows for the difference between 1995 and 2005.

7.3.7. Plotting and R Connectivity

Plots allow visualizing properties of data sets and correlations among attributes that are neither visible on the map nor recognizable in the data table. Vat has native support for some plot types, e.g. histograms. Here, Wave uses D3.js to visualize the graphs. Since plots are the result of an operator, users can use similar input dialogs to create them.

In addition to the native plots, Vat provides an interface for the programming language R. On the one hand, this allows experienced users to bring in their existing

[4]sedac.ciesin.columbia.edu/data/set/gpw-v4-population-count-rev10 (accessed Sept. 9, 2018)

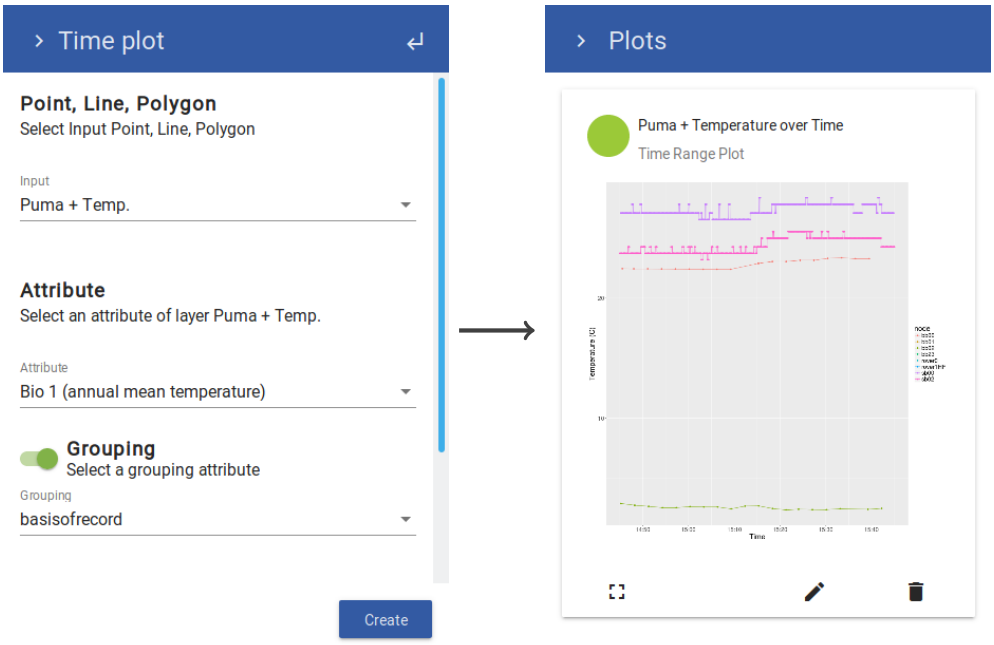


Figure 7.9.: This graphic depicts the application of a time series plot.

scripts, e.g. for sophisticated data statistics or for generating plots. On the other hand, we utilize the R interface ourselves to exploit its powerful visual components. For certain plot types, we provide a transparent interface that makes the functionality accessible to users without knowledge of R. Altogether, it is not necessary to reimplement certain functionality again. Among others, we have implemented scatter plots with trend lines, bar charts and temporal line charts that rely on the R connectivity. The last one allows plotting a time series for layer attributes and a time interval in the spatio-temporal query rectangle. Figure 7.9 shows the input dialog and one exemplary output for this plot.

VAT establishes a steady connection to an R server component. To accomplish this, our MAPPING back end links to an RServer[5] instance. We use remote procedure calls to execute code, send input data and return the finished script output. One problem is the impedance mismatch of our C++ and GPU-friendly data structures, which constitute, inter alia, in a columnar data structure for vector data. We use RInside [EF13] to implement a two-way mapping of all our data structures to equivalent R data structures. For this, we used the *sp* package [PB05] that provides geo data structures like data frames with spatial properties. In the R environment, VAT provides a custom namespace with specific functionality to access input data

[5] github.com/bgweber/RServer

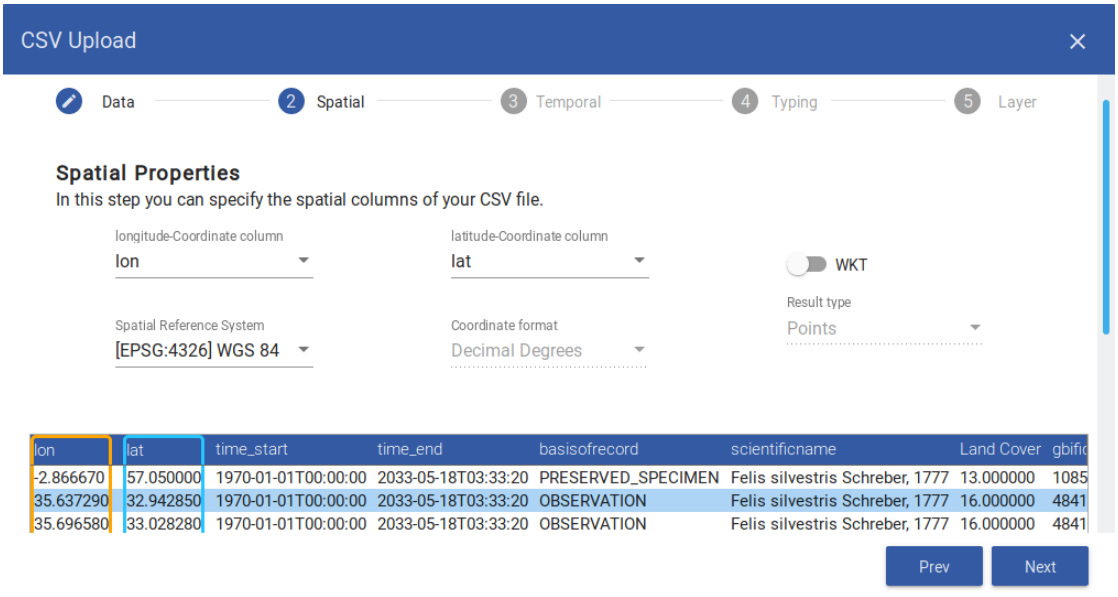


Figure 7.10.: This graphic shows an exemplary step of the CSV import dialog.

sets and the spatio-temporal query rectangle. The output data types are either raster or vector data, plots or simply texts. For security reasons, we make use of sandboxing to prevent the execution of malicious and system-threatening code on VAT's host system.

7.3.8. Data Import and Export

WAVE provides functionality to import custom data into the VAT System as well as to export results for publication or further usage. Exporting allows using the results, for instance, in custom applications or specific analysis tools. Furthermore, users can share results with each other.

For importing custom data, VAT implements various adapters to read basic vector data formats, e.g. CSV, and various raster data formats, e.g. GeoTIFF. Figure 7.10 shows the CSV import dialog of WAVE, which guides the user through the upload of a custom CSV data file. It is a multi-step approach:

1. The user uploads a CSV file and the dialog uses a data sample for configuration purposes.
2. The dialog automatically detects CSV delimiters, but lets the user modify the choice.

3. The user specifies the spatial column(s) of the file. This can be, for instance, one latitude and one longitude column for one point per line or a Well Known Text (WKT) [Ope10a] string that specifies a single polygon per line. The data projection is also configurable here.
4. The user specifies columns for the temporal validity. This can be, for instance, a start and an end column or a start and a duration column, e.g. in seconds.
5. The user can change the settings of the column data types of the remaining columns.
6. The last dialog step contains settings for error handling. It allows specifying the behavior in the case of erroneous lines. For instance, Vat can stop the whole import process or simply skip these lines. As another alternative, Vat can change the value of erroneous fields to default parameters. Furthermore, the user can change the name of the data set in this dialog step.
7. Vat imports the file, adds it to the map and provides a list of user uploaded data sets.

For raster data, Vat uses primarily the GDAL library, which provides connectors to a large number of data formats. In addition, Vat is able to integrate data from remote sources that provide standard interfaces such as WFS and WCS. For example, the US government provides such a service[6] for public data like educational and police reports. This allows the incorporation of a large variety of data sources and types into the Vat System.

Besides data import, Vat provides means to export the data out of the system again. Wave provides an export interface that starts a download of a selected layer, which is a result of a workflow computation. A data export always consists of a package (zip file) of three items:

1. the data itself, e.g. a raster in GeoTIFF or a vector data set in GeoJSON [But+16] format,
2. the workflow description for which Vat creates a JSON file that provides the computational graph and operator parametrization, and
3. the list of citations, which is again a JSON file that contains a list of citable items with the fields *citation string*, *license* and *URI*.

[6] www.data.gov

This emphasizes that the idea of Vat is not only to create a result, but also to always maintain the data lineage in form of a processing workflow and data source information. This makes the Vat System especially valuable in science.

For sharing results with other users, Wave allows creating a URL string that users can send to their peers. When users follow this link, Wave opens up and provides the options either to integrate a new layer with the shared workflow into their project or to start a project with the shared result only. Users are then able to review the workflow and citations of the computation and also integrate this result into their own, new computations. In the future, we aim to implement identifiers like the Digital Object Identifier (DOI) [Pas02] that would make it possible to provide links to results from a publication to a long-term instance of the Vat System.

7.4. Integration to Infrastructure Projects

This section briefly discusses the integration of the Vat System into a project environment. As a role model, we will first discuss the connection to the GFBio project. Then, we point out opportunities for using Vat in other infrastructure and research projects.

7.4.1. Connection to GFBio

The Vat System is an integral part of the GFBio (cf. Section 1.1) portal. By using this portal, researchers can create a data management plan for their research projects and at some point, at latest at the end of the project, submit their data to one of GFBio's associated partner data centers or archives. The other end of the spectrum concerns the reusage of this research data. Scientists who want to visualize or incorporate previously generated data can use the portal search to look up relevant data sets, review their metadata and download them [Löf+17]. Within GFBio, there exists an agreed set of mandatory metadata fields, e.g. spatial and temporal metadata that is available for every data set. The Vat System utilizes these fields to integrate respective search results.

The Vat System provides access to GFBio search baskets. Like in a warehouse, GFBio portal users can search for relevant data and put multiple data sets into a basket for further usage. Users can either follow a link from the portal or query the search basket from within the Vat System to access these data sets. Here, Vat accesses the data centers and archives via both customized and standardized

connectors. Examples are connectors to the Pangaea Data Warehouse [Pas02] and to the BioCASe providers [GBM07], which make data available in the XML-based ABCD [Acc07] standard. This leads to a close portal integration where users can easily transfer their search results to the Vat System.

In order to link GFBio portal users to the Vat user database, we established a single sign-on for GFBio users. When using Vat from a link from the GFBio portal, they will be automatically logged in. Otherwise, users can use their GFBio credentials to log into the Vat System. In the background, this describes an add-on module for the Vat user database. Thus, for a running Vat instance, users can use either their GFBio credentials or their Vat credentials to login and share their work.

7.4.2. Opportunities for Other Projects

The Vat System is not inherently integrated into the GFBio infrastructure. On the contrary, it is based on a module system. Thus, it can principally offer the same functionality to other research and infrastructure projects. For instance, we established a cooperation with the GEO BON[7] project to create an EBV Spatial Analyzer [Bei+18] as a dedicated web application. EBVs (Essential Biodiversity Variables) [Per+13] present derived measurements that groups of biodiversity researchers agree upon and which are important indicators for the world's biodiversity. Forest gain and loss models are one example, whereas the majority of variables still needs to be developed or decided on. Here, we provide a Vat instance with a modified user interface based on Wave. It uses pre-defined workflows that are created in a first step with the basic system. In a second step, the resulting workflows are visualized in a report version with only few options or parameters to change. For instance, one could visualize the forest change together with a time series plot and provide the user only with the possibility to change the time period.

By maintaining the modular structure of all Vat components, it is possible to also provide adapted versions to other research projects. This allows using the powerful core functionality of the Vat System and to change only the look and feel as well as the options for the user. This would facilitate a faster development of such applications and the use of joint forces to continue developing the Vat System.

[7] geobon.org

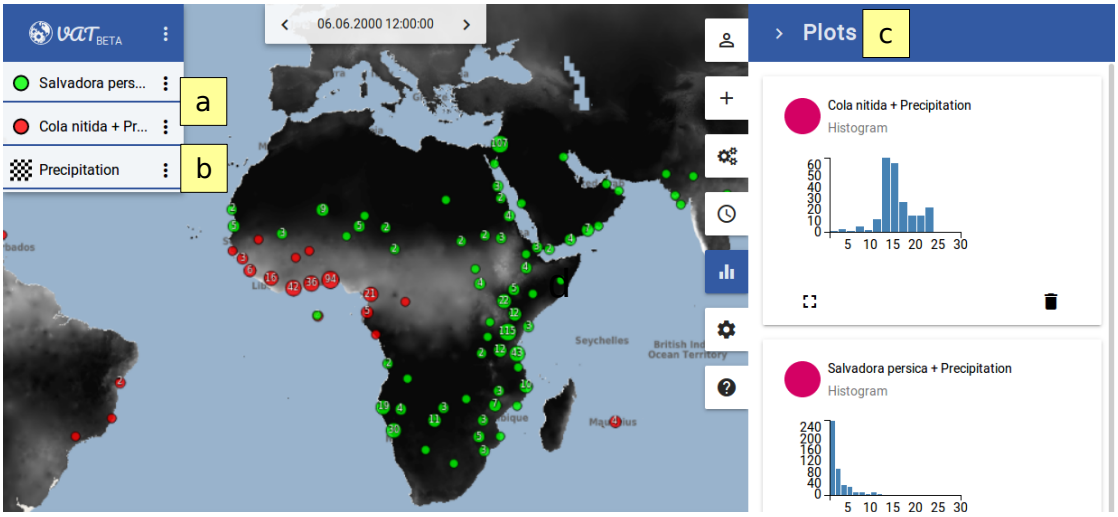


Figure 7.11.: This screenshot presents the final stage of the use case, in which markers indicate the different steps that have led to the results [Bei+17b].

7.5. Example Use Case

In the following, we present a use case that exemplifies the capabilities and the usage of the VAT System. A public version of the system is available[8] and allows conducting this scenario. We consider the scientific task of discovering differences in the distribution of two tree species: the toothbrush tree (*Salvadora persica*) and the kola nut (*Cola nitida*). While the toothbrush tree is a small evergreen tree that grows in hot and dry conditions, the kola nut is a tropical tree from West African rainforests. Our use case includes processing and visualizing a combination of raster and vector data, comparative plotting and time shifting.

In a first step, occurrence data from both species is added to WAVE. Users often want to combine their own data with existing public data sets. Thus, the use case contains one user data set and two data sets from one of the repositories provided by VAT. The user data is given as a CSV file and is therefore imported with the import wizard from WAVE's data menu. Having specified the data format as well as spatio-temporal properties, the data set is available in the private repository of the user and added as a layer to the map. Then the species occurrence wizard from WAVE's data menu is used. Here, the kola nut is looked-up and the resulting GBIF data is added as layer to the map.

[8] vat.gfbio.org

Figure 7.11 shows the map with layers for both species (cf. marker (a)). Their distributions are matching their habitat descriptions. The kola nut occurrences are in rainforest areas while toothbrush trees are located in hot and dry regions. To confirm the visual impression, the user adds environmental data to the project.

In the data repository of Wave's data menu, monthly precipitation data is available from the WorldClim data set. Adding it as a new layer allows assessing the precipitation at the occurrence points by visual inspection. Now the *raster value extraction* operator from the operator menu allows adding the raster value at each occurrence location to the occurrence points by joining the two data sets. This creates a new layer where all points have an additional attribute *precipitation* that reflects the average monthly precipitation at the location of the point.

Figure 7.11 shows layers for both species and the precipitation (cf. marker (b)). In order to compare the two data sets, the histograms of the precipitation at the occurrence locations are generated with the histogram operator from Wave's operator menu. Here, both histograms use the same configuration (e.g. a range of 0 mm to 300 mm and 20 buckets) to achieve comparable results.

Figure 7.11 shows the histograms of precipitation values at the occurrence locations of the kola nut and the toothbrush tree (cf. marker (c)). We observe that kola nut occurrences are more often located in areas with high precipitation rates and toothbrush trees in dry areas. The precipitation data from WorldClim represents monthly aggregates. Wave's temporal reference toolbar allows traversing the data month-by-month by changing the displayed point in time. By changing the reference time in Wave, the layers on the map, the values in the data-table and the plots are re-generated automatically. For creating sophisticated species distribution models on the user's personal computer, the enriched species layers are exported as CSV files by using the export dialog found in the layer's context menu.

7.6. User Interface Design

An intuitive user interface is of utmost importance to the Vat System's ecosystem. The goal was the realization of the concept of an interface for exploratory data analysis and time series data computations. The development of Wave was conducted in two phases.

7.6.1. The Two-Phase Approach

In the first phase, we discussed many design decisions on a white board. We combined our ideas into a paper prototype, which was iteratively improved. While the first paper prototypes were on real paper, later versions were developed using software tools like balsamiq[9] and inVision[10]. This allowed us to easily incorporate button clicks and multiple page views, and to present them to users. This phase contained a user evaluation with domain experts. We will discuss the results in more detail in Section 7.6.2. With these results, we reworked the paper prototype and finished phase one.

In the second phase, we started implementing the actual web application. For this, we first took our conceptual ideas from the paper prototype into account for choosing the most promising web technologies and frameworks in order to accelerate the implementation process. As discussed earlier, we mainly rely on Angular and its Material Design implementation, which we started using as early adopters in the beta phase. Fortunately, as it is developed mainly by Google Engineers, there are stable and (which is unusual for web frameworks) long-term stable versions available now. When the first working version was available, we conducted a second user evaluation, which we will also discuss in Section 7.6.2. As in the first phase, the results led to some modifications of the user interface. Since the development of the paper prototype was evaluated thoroughly and provided a blueprint, there were no immense redesign steps necessary. One noticeable design change was, for instance, the menu bar's switch from the top to the right side of the screen. This provided more screen space, particularly on mobile devices such as tablets, but also on widescreen monitors.

After these two steps, we started integrating the VAT System into the GFBio portal, while simultaneously improving it and extending its functionality. For maintenance, regular portal user tests ensure that updates in the GFBio ecosystem do not interfere with the interaction of the VAT System with other components. For this, the GFBio consortium implemented several user stories and use cases that are manually tested on releases of new versions.

7.6.2. User Evaluation

To evaluate the usability of WAVE we conducted both user studies with biodiversity experts from the Senckenberg Biodiversity and Climate Research Institute (BiK-

[9] balsamiq.com
[10] www.invisionapp.com

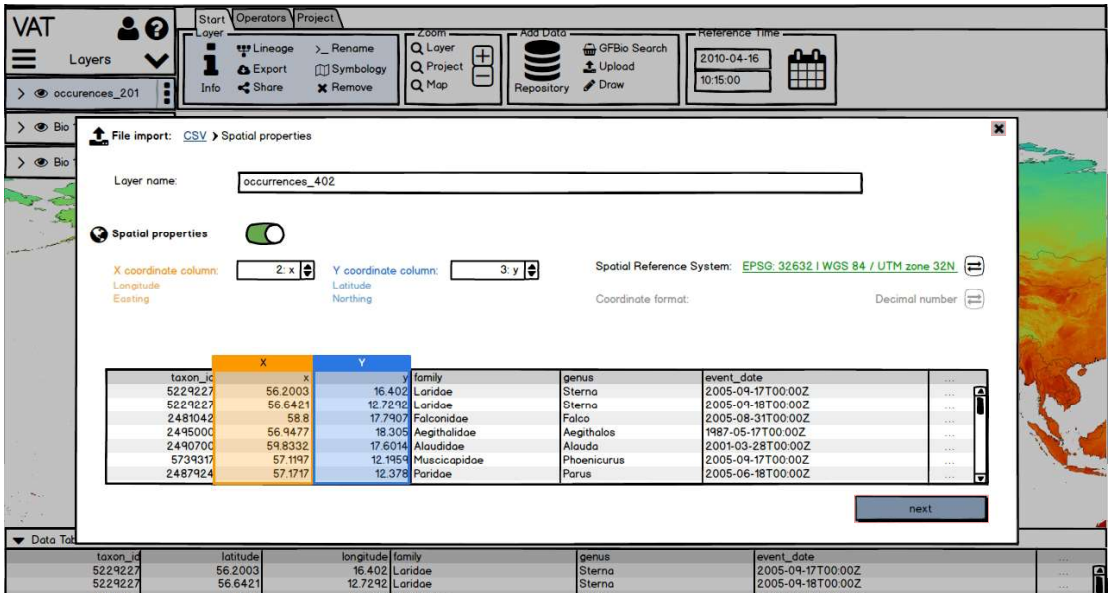


Figure 7.12.: This graphic shows the CSV upload dialog of the paper prototype [Bei+17b].

F) in Frankfurt am Main, Germany. The first user study took place in the design phase of WAVE and was conducted to gain insights about an appropriate user interface design prior to the product development. The second user study was designed to evaluate the actual implementation of WAVE and featured the use case provided in the previous section.

Conceptual Design Evaluation

The conceptual design evaluation included 15 potential users in the design process of creating an effective user interface. For this we developed a paper prototype (cf. Figure 7.12), which covers an almost identical use case to the one described in the previous section. The use case focuses on bird occurrences instead of tree occurrences, but uses the same operations. This allowed us to abstract all implementation details and to focus on concepts on a sketch board. The advantage was that it was very inexpensive to discard inadequate concepts. And in conclusion, this led to rapid concept development with domain experts.

The user study consisted of two parts. The first one consisted of an introduction to the use case and a 20-minute time span to solve a specific task. The users had to work on the task independently without any system introduction or explanation. We observed their behaviors and timed their steps doing certain sub tasks. The

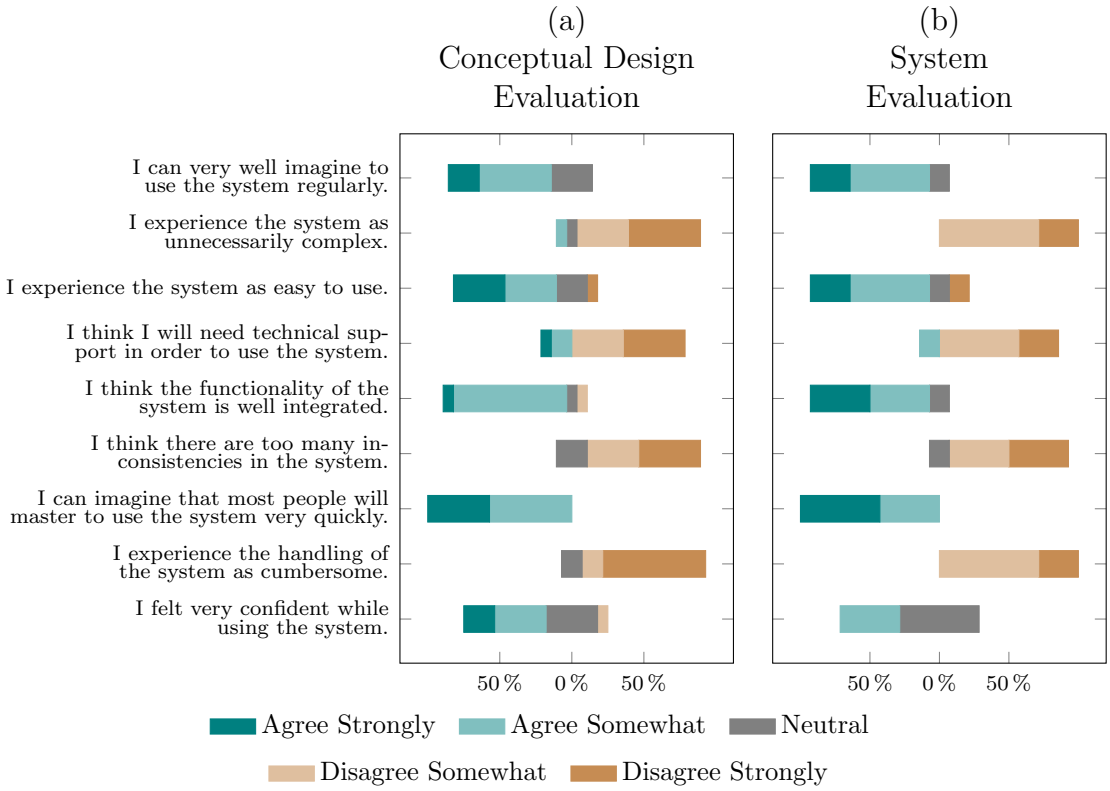


Figure 7.13.: These bar diagrams show the results of the two user evaluations [Bei+17b].

second part included a questionnaire of nine fixed questions and an additional field for free text comments. The participants had ten minutes time for their feedback. The questions aimed at different impressions about the system usage. The users were asked to choose from typical symmetric five-level Likert scaled answers with a neutral element (cf. Figure 7.13 (a)).

Our analysis combined the behavioral observations, the answers to the questionnaire as well as additional comments. In conclusion, we did not find any reason for major changes in our design proposal. Nevertheless, we identified minor weaknesses and were able to get a better understanding of how users intend to work with our system. One interesting fact to mention was the users' expectation to interact with the application in a similar way as with desktop GIS. This included right-clicking on elements to perform actions. This was a strong contrast to our previous experience in web application development. In addition to the feedback from the first user study, we were asked by early testers to move the toolbar from the top to the side of the screen (cf. Figure 7.11). The idea was to utilize

the typical wide-screen aspect ratio of modern desktop and mobile displays more efficiently.

System Evaluation

The second user study included seven participants. While the evaluation design was the same as in the first evaluation, the participants used an early implementation of Vat and not a mock-up. The equal experiment design allows a direct comparison of both study results. First, we introduced the use case that focused on toothbrush tree and kola nut (cf. Figure 7.11). Then, again, the users worked independently without introduction or explanation of Wave to solve the tasks. In comparison to the first study, we extended the time span to 30 minutes. None of the users had a previous experience with using Wave. We monitored the screens of the participants and their mouse movements. Additionally, we asked them to express their thoughts (think-aloud method [Jas+04]) and recorded everything in detail in writing. After finishing their tasks, the participants filled out the same survey we used in the course of the first evaluation to assess Vat (cf. Figure 7.13 (b)).

In summary, the results were very positive. The participants identified no major issue, so that no conceptual change of Wave was made. Minor disturbances in the workflow could be explained by the deviations from the participants' daily used tools. The temporal functionality, which is a novel feature in this domain, was very much appreciated. As the provided data sets were also highly appreciated, most participants asked us to extend the repository with more specialized data sets. Some users mentioned that they will most likely use Vat in the future and think that we should start providing Vat functionality for other biodiversity projects, e.g. for landing pages of data sets.

7.7. Related Work and Systems

This section presents related work and related systems that share common goals with the Vat System. We divide them into six groups, based on different aspects. For each group, we introduce the most prominent representative systems or research.

First of all, there are portals and catalogues for accessing geographical data. Examples are the Atlas of Living Australia [Bel11], LifeWatch [BL12], Integrated Digitized Biocollections (iDigBio) [Bea14] and the Catalogue of Life [JWO11].

These portals make data for several communities available, e.g. the Atlas of Living Australia provides environmental pre-aggregated data for the Australian continent. While these full-fledged solutions can be seen as competitors for the GFBio projects, Vat – as a part of GFBio – presents a more powerful geo processing engine that allows for dynamic processing of collected data. A specialized interface for Vat that hides all the complexity could bring up a similar front end to a data archive. CKAN [Win13] and GeoNetwork [TH07] are two data catalogues. While these rely on third-party tools like GeoServer [Iac17] for data visualization, Vat rather compares to these third-party tools than to the catalogue itself. Vat can be seen as one building block that could be integrated in one of the previously mentioned solutions.

Second, there is geo software that aims for publishing data interactively on a map. The most prominent software is GeoServer [Iac17], which is the standard implementation of several OGC protocols [Ope07]. It supports various raster and vector formats as well as connections to databases with geo extensions, e.g. PostGIS [OH15]. While GeoServer constitutes a web service that has to be installed locally, GIS Cloud [LPB16] and CARTO [Zas15] are cloud-based services. GIS Cloud focuses on integrating custom data (mostly vector data) for an online, shared access, CARTO has its focus on producing beautiful maps. None of these systems provides sophisticated means for data processing. For CARTO there are some pre-defined analyses available and GeoServer can leverage the operators from the underlying database. Furthermore, GeoServer provides means to rasterize vector data. Vat also supports retrieving and publishing data using OGC standard protocols. While it has limited capabilities of map layouting and designing, it focuses more on geo data processing.

Third, there are workflow systems. Taverna [Wol+13] and Kepler [Alt+04] are classical examples for this system category, which offer users an interface to define a processing workflow upfront. The execution of a workflow leads eventually to the output of the result. Pegasus [McL+13], the Google Earth Engine [Gor+17] and OmniSci [Mos17] (formally MapD) are modern representatives that focus on distributed processing. Especially the Google Earth Engine incorporates the machine learning framework Tensorflow [Aba+16] to incorporate state-of-the art algorithms in geo processing. OmniSci incorporates in-memory processing with GPUs to achieve a high throughput. However, Vat is designed to give users an interactive experience and defines workflows along the way of exploring the data. Hence, it uses a similar paradigm, but approaches it from another perspective. While Taverna and Kepler provide the user at least with a graphical user interface, Pegasus and the Google Earth Engine require programming the workflow. There are special-purpose systems like the Map of Life [JMG12], which provide

a front end to answer distinct questions in the domain of biodiversity. They preprocess their data with the Google Earth Engine and deliver the results to the user.

Fourth, there is standard geographical information system (GIS) software. A prominent open source representative is QGIS [Gra13]. In general, GIS offer a graphical user interface that requires only little programming skills. Desktop GIS, however, have some disadvantages. First, they suffer from slow processing as they are limited to local resources and do not exploit modern hardware sufficiently well [Aut+15b]. Second, to the best of our knowledge so far no GIS exists that treats every data set as a time series and produces derived data sets with valid temporal information dynamically on demand. Existing GIS like GRASS [Peb17] have extensions that allow processing temporal data sets. However, to match their existing processing model, they process the data upfront rather than on demand. Moreover, they store intermediate steps as new data sets on disk, which leads to a high consumption of storage space and slow response times for exploratory data analysis.

Fifth, in contrast to desktop GIS, Zhang et al. [ZYG17] are developing a web-based GIS for exploratory usage. They incorporate GPU-accelerated processing similar to the VAT System. Their work focuses on extensive indexing support for evaluating point-in-polygon predicates. However, they base their front end on the Google Maps Platform [Pet15] for performance and ease of use, but focus mainly on the back end part. In contrast, we consider a good user interface to be equally important as a high-performance back end. Thus, we spent particular effort on accomplishing a high level of usability. We conducted extensive user studies to validate our user interface design and identify opportunities for improvements.

Sixth, there are systems that provide either raster or vector processing. RasDaMan [Bau+98] and SciDB [The10] are the most prominent examples for raster processing in the form of efficient array computing. On the other hand, there are efficient vector processing systems like SparkGIS [Bai+17], GeoSpark [YWS15] and STARK [HGS17] that are all based on Apache Spark [Zah+10]. Pandey et al. [Pan+18] investigated various Spark-based systems in terms of their functionality and performance. Unlike VAT, they do not provide combined geo processing with raster and vector data, but only support a subset of vector data types. However, as they focus only on vector data, they offer more possibilities with respect to join predicates, e.g. k-nearest neighbor joins. Beyond that, all systems are server-side only and do not inherently provide a powerful user interface.

From our experience, expert users do not always rely on GIS and often program

their workflow in their programming environment. The programming language R is an often-used candidate since it offers large statistics functionality without the necessity to use third party libraries. Systems like the Berkeley Ecoinformatics Engine [Rap+14] serve as storage for vectorial biodiversity data and environmental raster data. In this case, it provides an API to query the data with filter predicates. However, it does not provide sophisticated processing functionality itself. In Vat, users can augment the system by using custom R operators. Instead of downloading all data to the local machine that runs R, we allow uploading missing operators to the processing environment. Thus, users can rely in most cases on our high-performance operators and only have to switch to custom operators if really necessary.

A cross-cutting concern for all geo processing in science is the generation of proper citations [AK07]. Since it is often difficult to convince researchers to publish their data alongside with their findings if the funding agencies do not require it, it is of utmost importance that new research gives sufficient credit to the data producers. Automatic citation generation is key to this question. Buneman et al. [BDF16] tackle this task for database systems in general. The idea is to subdivide the input data into citable units. When citations need to be specified, citable units represent the most fine-grained levels to look at. Their method uses common query rewriting techniques to transform the query into views such that each view corresponds to a citable unit. If the view is still in the re-written query, the data's metadata is part of the citation. While the problem of query re-writing is NP hard in general, there are applicable heuristics to use. Other solutions [Rau+16] also try to find the best subsets of the data that form the citations. Since Vat uses custom operators for each processing step, the citation tracking is integrated there. This means that it is not necessary to rewrite the queries afterwards.

7.8. Summary

This chapter presented the Vat System, which is an interactive web-based geographic information system for exploratory data analysis of spatio-temporal data. We showed the novelties of the system that includes inherent time series support as well as lineage and citation handling. In addition, we showed that data aggregation techniques like the Circle Merging Quadtree (CMQ) were adapted to be used within the system. These techniques foster exploratory analysis. Besides that, the integration of R allows us reusing existing functionality, e.g. for plots, and integrating user-defined functions.

We additionally showed the connection to the GFBio project. Besides the inclusion of various generic data sources, the seamless connection to the project's portal encourages data reusage of biodiversity research data. Taking GFBio as a role model, this integration advocates the utilization of the Vat System for other research projects.

Furthermore, we presented the conceptual design and implementation of the user interface (Wave) of the Vat System. Here, we discussed two user studies that facilitated design decisions along the way. As a result, Wave promotes the exploratory usage and harmonically transforms the user's actions into queries to the back end server (Mapping).

Finally, we presented related systems of different categories and discussed their relationship with the Vat System. In this context, we identified similarities and distinguished ourselves from their methodology. As a result, Vat is a system that addresses geo processing for exploratory research in a novel way.

8 Conclusion

This final chapter concludes this thesis by first giving a summary of the presented content. Here, we point out the most important findings and put them into context. Then, we point out ideas for future research in the field of visual point clustering and exploratory geographical information systems.

8.1. Summary

In this thesis, we addressed the problem of visualizing large spatial point data sets. We defined the problem of visual point clustering that leads to proportional circle maps. These maps represent aggregates of the underlying data sets that provide important information about distribution, cardinality and locality.

For investigating the quality of a visual point clustering result, this thesis introduced a set of quality measures that address different aspects of a good circle representation. In this sense, we investigated generic clustering methods for their suitability in this problem setting. The findings showed that there were acceptable results possible in principle, but the parametrization poses a problem that makes it infeasible to use these algorithms.

Moreover, this thesis presented the Circle Merging Quadtree (CMQ). This method constitutes a novel and efficient approach for visual point clustering. Since its basic form does not provide computational stability, we extended the algorithm by a preprocessing that solves this issue and substantially improves the overall runtime. The experimental evaluation showed that CMQ outperforms related algorithms in terms of runtime. Additionally, we demonstrated the high quality of its results by evaluating the quality measures on several real-world data sets from the field of biodiversity. Moreover, the aggregation of points to circles leads to, in comparison, very small output data sets that are suitable for mobile devices.

This thesis furthermore showed extensions to the basic *CMQ* method with respect to miscellaneous attributes and to multiple data sets or classes within a data set. For miscellaneous attributes, we detailed on suitable aggregation methods that allow to efficiently compute statistical measures that are suitable for side-views like plots or data tables. For multiple data set or classes, we extended two methods from literature, pie chart maps and maps with circle packings, to work with *CMQ*. For the latter variant, we proposed several improvements that were also evaluated within a user study. The study revealed the positive effect on the visualization of multiple data sets that is provided by this *CMQ* extension.

Since *CMQ* is a method that is ideally suitable for visual analytics systems, we embedded it into the context of an exploratory geographical information system. The Visualization, Analysis and Transformation System (Vat) constitutes such a visual analytics system for spatio-temporal data. Here, we detailed on the development of an interactive user interface that facilitates exploratory research. Two user studies verified the success of the development of Vat.

Overall, this thesis provided significant contributions in the field of spatial data visualization and spatio-temporal visual analytics systems for research that incorporate geographical data.

8.2. Future Work

For future research, we discovered three possibilities. First, we discuss aggregations with further topological constraints. Second, we propose the parallelization of *CMQ*. Third, we open up the field of stream processing in the context of visual point clustering.

8.2.1. Aggregation with Topological Constraints

The visual point clustering leads to aggregating data of multiple nearby locations to new circles in a location (and with a size) that represents all of these included locations best. However, there are reasonable arguments for scenarios in which this leads to undesired effects. For example, points of observations of sea animals in combination with points of terrestrial animals can potentially lead to circles in the ocean that represent the terrestrial species, or vice versa. Another example is data that is gathered from different countries. In this case, we

might want to provide an aggregated view that does not merge data from adjacent countries.

We can achieve these topological constraints by incorporating lines or polygons that act as boundary objects [Gru+15]. Hence, we would forbid merging two or more objects across those boundaries. A possibility is to introduce a data structure for boundary lookups, e.g. another quadtree. Each merge would then be preliminary and checked for an overlap with a boundary by querying this data structure. If an overlap occurs, the merge would then be reverted.

It may be necessary to introduce displacements in the constrained aggregation case. If a circle crosses a boundary, it gets *pushed back* in order to eliminate the overlap. Note that displacements could lead to potentially new overlaps with circles that were previously unaffected.

8.2.2. Parallelized CMQ

In the chapter about the VAT System (cf. Chapter 7) we discussed the utilization of GPUs in order to speed up computations by exploiting a high degree of parallelism. By now, although being very efficient, the visual point clustering computation of *CMQ* is linear and single threaded. However, we see great potential to improve *CMQ* even further by enhancing certain steps with parallel computation.

The computation of the preprocessing step incorporates a grid data structure. This correlates to the rasterization of points by introducing additional constraints. For instance, our grid constitutes a (very) large spatial histogram. Common GPU-based histogram computations [Góm+13] store one histogram for every computational group in local memory. This, however, quickly exceeds the memory of modern GPUs in the case of large histograms. On the other hand, one can generally iterate over all points in parallel and access the grid cells concurrently. Unfortunately, a large number of threads that access a constant size grid leads to many concurrent accesses and, hence, locking and synchronization. This means it is not possible to achieve a highly parallel implementation this way. A compromise pose adapted group prefix-sum (grouped scan) operations [KML15]. An experimental evaluation can then reveal the best strategy with respect to certain degrees of parallelism, size of the map (and circles) as well as number of points as input.

Moreover, the computation of the insertions as well as the queries to the quadtree of *CMQ* can benefit from parallel computation [ZYG14]. The quadrants of the quadtree can be visited in parallel since partial results of the QUERYANDEXTRACT

method do not interfere with each other. Hence, with respect to the rather small depth of the tree, a certain degree of parallelism is possible here as well. Synchronization mechanisms allow combining the partial query results. Recall that we adapted the insertion such that it exploits the result of the queryAndExtract method on misses. It returns the path to the empty spot in the tree. Thus, there is no room for additional parallelism.

Since *CMQ* has a low memory requirement, it is possible to extend the algorithm to compute batches of input data sets that exceed the GPU's memory. In this case, either the resulting tree or the grid structure remain in the GPU's main memory. The remaining space has to be adequately subdivided between input buffers of point data and caches as well as local variables.

8.2.3. Streamed Clustering

Another use case for *CMQ* is monitoring a stream of spatial data [KK13]. Here, *CMQ* can constitute an efficient visualization that needs to be updated according to the incoming (and expiring) data in the stream. Depending on the type of data stream (complete history, tumbling or sliding [Li+05]) it is necessary to update *CMQ* differently. Two examples are the live observation of animal species or the city monitoring of parking cars.

If it is required to visualize the complete history, there is no adaptation necessary in order for *CMQ* to work. Due to the constrained space requirements of *CMQ* (cf. Chapter 5) there is no need to remove any data in this case.

For tumbling windows, it is sufficient to compute a visual point clustering for *CMQ* for every window. Since *CMQ* is efficient, its computation can take place alongside the processing of the streaming data. In this case, *CMQ* can output both intermediate results as well as the complete results at the end of each window.

For the last case of sliding windows, it is necessary to update *CMQ* such that it allows removing outdated or invalid points. For instance, we can only use the preprocessing grid and compute the remaining tree on demand or in fixed time intervals. If we know the elements to remove, we can undo the merge step by inversely calculating the weighted average. In the case that we do not know the elements to remove, we can compute *CMQ* for the data that falls out of memory. Then, the calculation of a weighted moving average [Fin09] between two *CMQ* structures has to be computed. However, suitable semantics for this computation and its validity has to be properly defined first.

An experimental evaluation can in either case show differences in updating the grid or the whole *CMQ*. Furthermore, one can assess the differences of the *static* version of *CMQ* and the streaming version in terms of quality. Additionally, *CMQ* can potentially act as a sketch [Fin09] of the data to provide further statistics other than plain visualization. The extent of this is also up to further research.

Appendices

A. Quality Measures Experiment

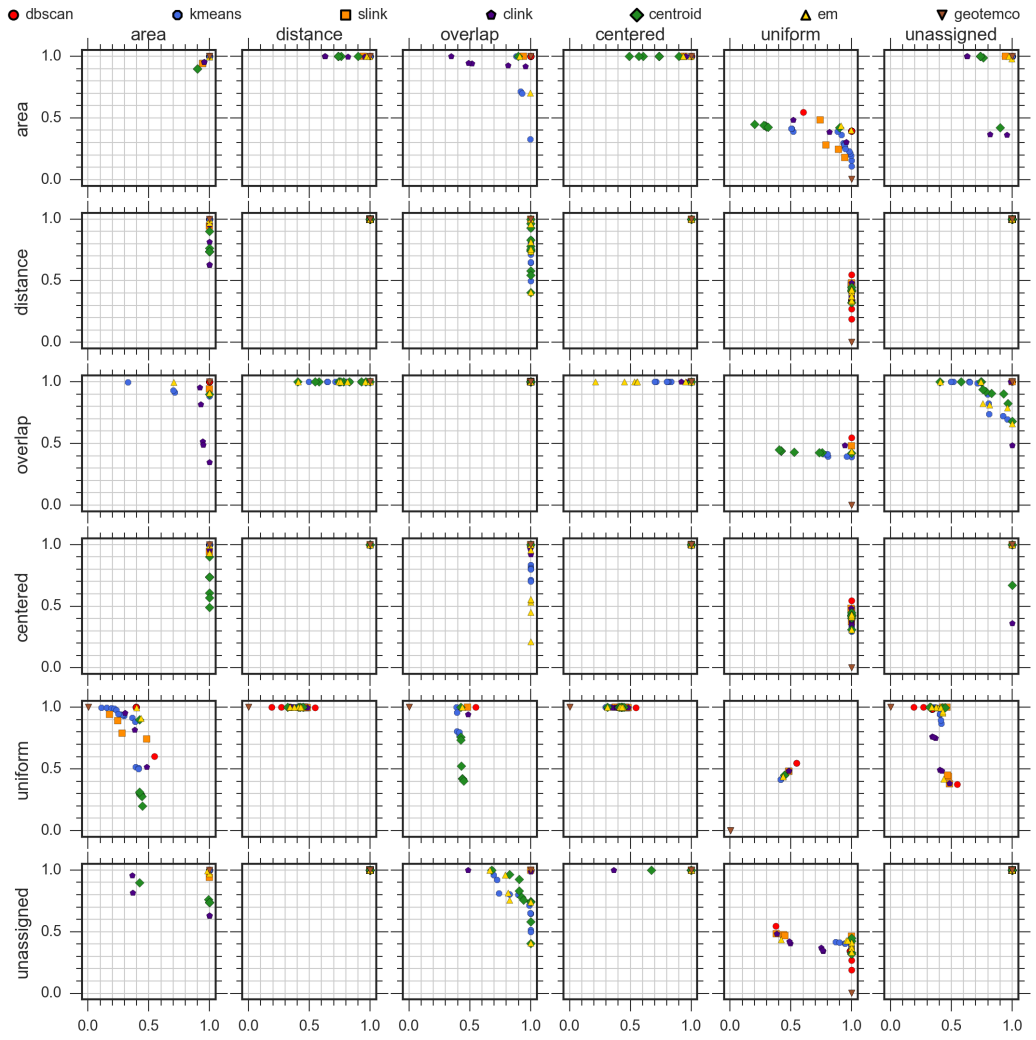


Figure A.1.: German libraries (3384 points)

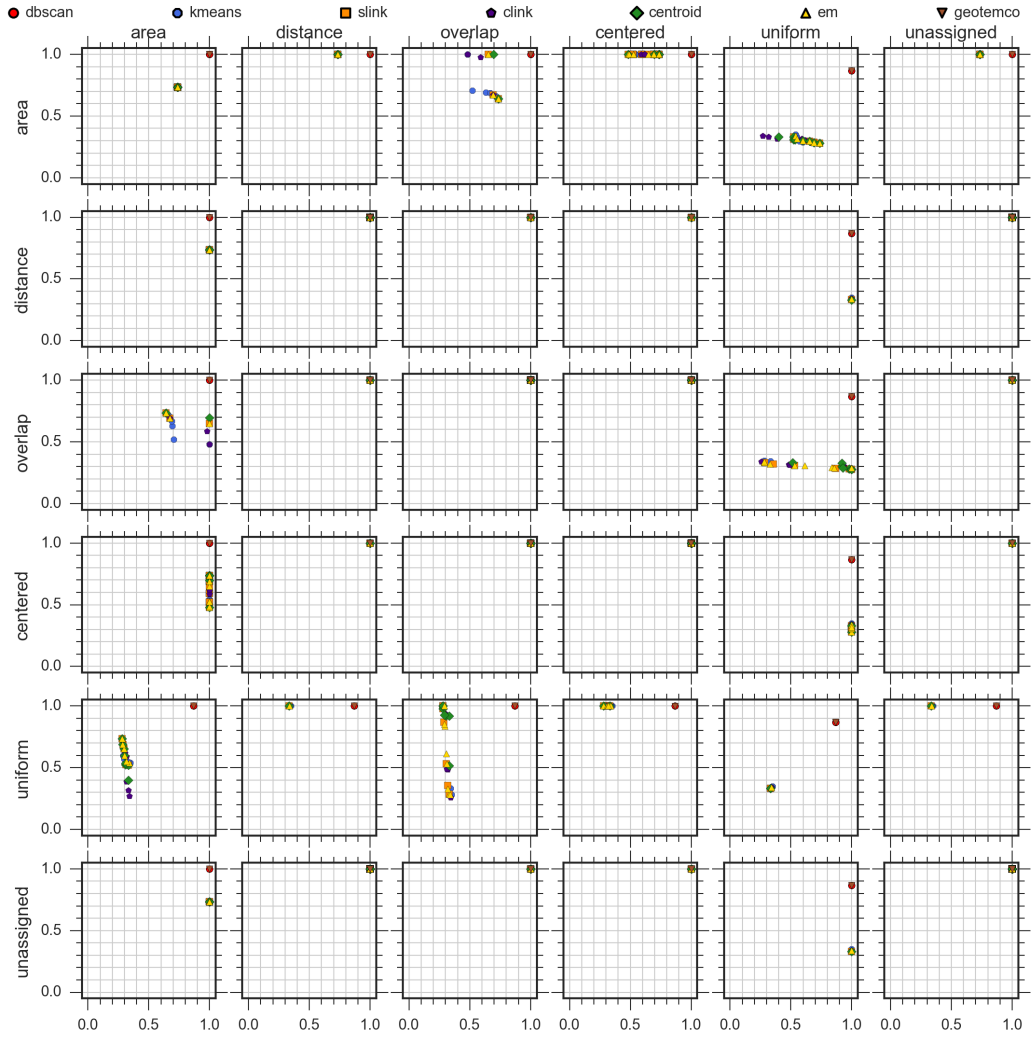


Figure A.2.: Loxodonta cyclotis (24 points)

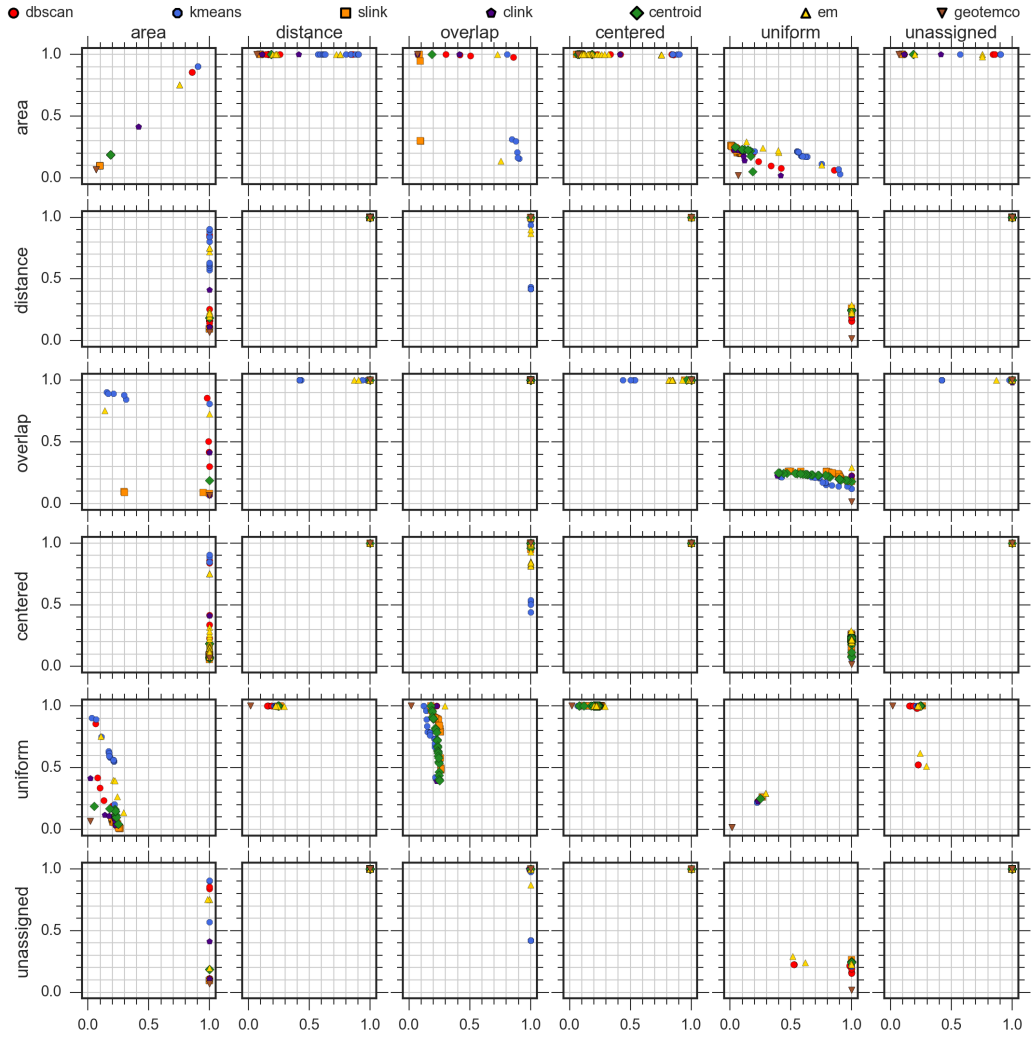


Figure A.3.: Macropus giganteus (23040 points)

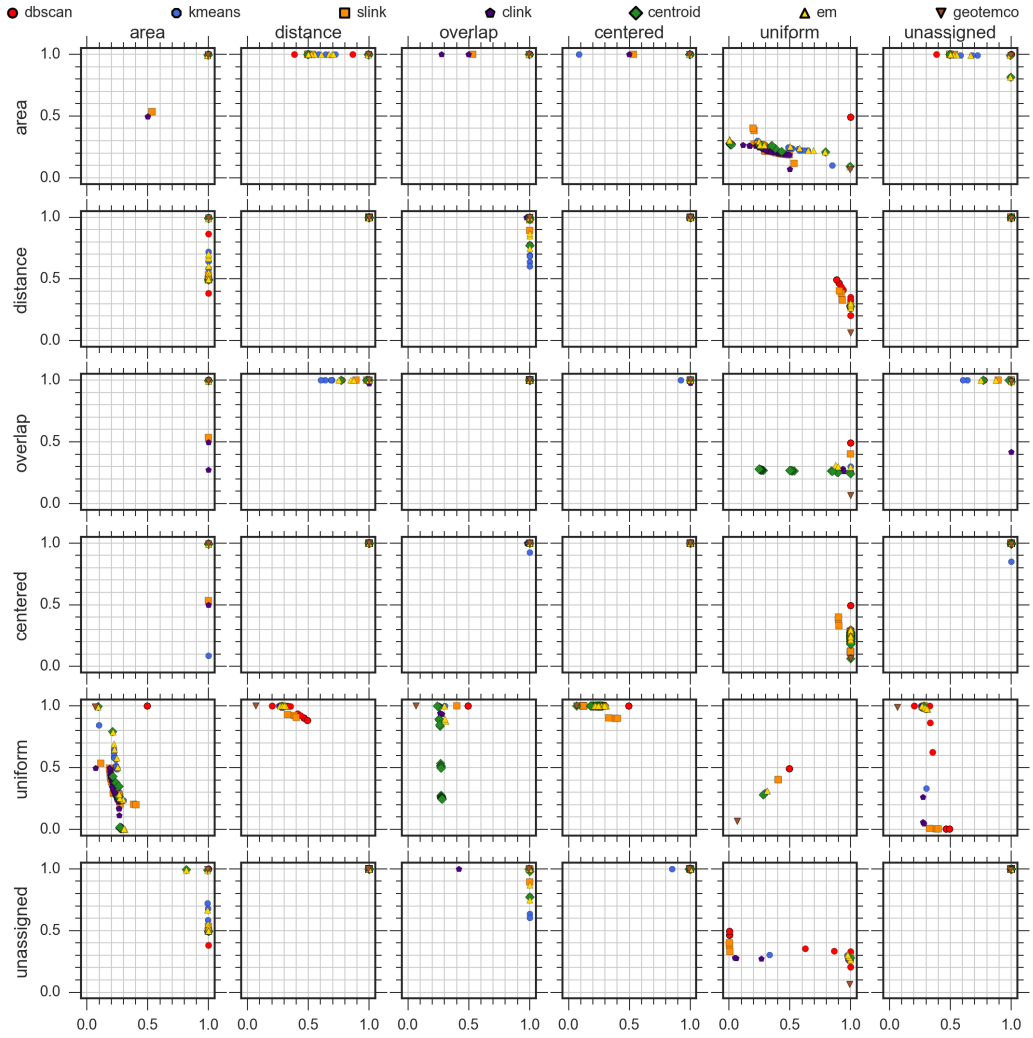


Figure A.4.: Puma concolor (1993 points)

References

[Aba+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, and G. Brain, "TensorFlow: A System for Large-Scale Machine Learning," in *OSDI '16: 12th USENIX Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283.

[Acc07] Access to Biological Collections Data Task Group, "Access to Biological Collection Data (ABCD), Version 2.06," *Biodiversity Information Standards (TDWG)*, 2007.

[Ach+12] E. Achtert, S. Goldhofer, H.-P. Kriegel, E. Schubert, and A. Zimek, "Evaluation of Clusterings - Metrics and Visual Support," in *Proceedings of the IEEE 28nd International Conference on Data Engineering (ICDE)*, Washington, DC, USA: IEEE Computer Society, 2012, pp. 1285–1288.

[Ack89] R. L. Ackoff, "From Data to Wisdom," *Journal of applied systems analysis*, vol. 16, no. 1, pp. 3–9, 1989.

[All83] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[Alt+04] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, IEEE, 2004, pp. 423–424.

[AK07] M. Altman and G. King, "A Proposed Standard for the Scholarly Citation of Quantitative Data," *D-Lib Magazine*, vol. 13, no. 3/4, 2007.

[Alu04] S. Aluru, "Quadtrees and Octrees," in *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sahni, Eds., Chapman and Hall/CRC, 2004, pp. 19–1–19–26.

[And73] M. R. Anderberg, *Cluster Analysis for Applications*. New York, NY, USA: Academic Press, 1973, p. 372.

[And+10] G. Andrienko, N. Andrienko, U. Demsar, D. Dransch, J. Dykes, S. I. Fabrikant, M. Jern, M.-J. Kraak, H. Schumann, and C. Tominski, "Space, Time and Visual Analytics," *International Journal of Geographical Information Science*, vol. 24, no. 10, pp. 1577–1600, 2010.

[AAG03] N. Andrienko, G. Andrienko, and P. Gatalsky, "Exploratory Spatio-Temporal Visualization: An Analytical Review," *Journal of Visual Languages and Computing*, vol. 14, no. 6, pp. 503–541, 2003.

[ASS02] L. Anselin, I. Syabri, and O. Smirnov, "Visualizing Multivariate Spatial Correlation with Dynamically Linked Windows," *Urbana*, vol. 51, p. 61 801, 2002.

[Arb93] G. Arbia, "The Use of GIS in Spatial Statistical Surveys," *International Statistical Review*, pp. 339–359, 1993.

[AV07] D. Arthur and S. Vassilvitskii, "K-Means++: The Advantages of Careful Seeding," in *SODA '07: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[AKL13] F. Aurenhammer, R. Klein, and D.-T. Lee, *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

[Aut+15a] C. Authmann, C. Beilschmidt, J. Drönner, M. Mattig, and B. Seeger, "Rethinking Spatial Processing in Data-Intensive Science," in *BTW 2015: Datenbanksysteme für Business, Technologie und Web - Workshopband*, vol. P242, Bonn, Germany: Gesellschaft für Informatik e.V., 2015, pp. 161–170.

[Aut+15b] C. Authmann, C. Beilschmidt, J. Drönner, M. Mattig, and B. Seeger, "VAT: A System for Visualizing, Analyzing and Transforming Spatial Data in Science," *Datenbank-Spektrum*, vol. 15, no. 3, pp. 175–184, 2015.

[Bad13] M. Bader, *Space-Filling Curves - An Introduction with Applications in Scientific Computing*. Berlin, Heidelberg, Germany: Springer, 2013.

[Bae+07] W. D. Bae, P. Vojtechovsky, S. T. Leutenegger, S. Alkobaisi, and S. H. Kim, "An Interactive Framework for Raster Data Spatial Joins," in *GIS '07: Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*, New York, NY, USA: ACM, 2007, pp. 1–7.

[Bai+17] F. Baig, H. Vo, T. Kurc, J. Saltz, and F. Wang, "SparkGIS: Resource Aware Efficient In-Memory Spatial Query Processing," in *SIGSPATIAL '17: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 2017, 28:1–28:10.

[BL12] A. Basset and W. Los, "Biodiversity e-Science: LifeWatch, the European Infrastructure on Biodiversity and Ecosystem Research," *Plant Biosystems*, vol. 146, no. 4, pp. 780–782, 2012.

[Bat+14] S. E. Battersby, M. P. Finn, E. L. Usery, and K. H. Yamamoto, "Implications of Web Mercator and Its Use in Online Mapping," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 49, no. 2, pp. 85–101, 2014.

[Bau+98] P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann, "The Multidimensional Database System RasDaMan," in *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 1998, pp. 575–577.

[Bea14] J. H. Beach, "Conceptualizing and Managing Paleontological Collection Data with Specify Software," in *GSA Annual Meeting: Advancing the Digitization of Paleontology and Geoscience Collections: Projects, Programs, and Practices II*, 2014.

[Bec+90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," in *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 1990, pp. 322–331.

[Bei+18] C. Beilschmidt, J. Drönner, N. Fernández, C. Langer, M. Mattig, and B. Seeger, "Towards an EBV Analyzer based on VAT," in *ICEI 2018: Proceedings of the 10th International Conference on Ecological Informatics: Translating Ecological Data into Knowledge and Decisions in a Rapidly Changing World*, J. Gaikwad, B. König-Ries, and F. Recknagel, Eds., Jena, Germany, 2018.

[Bei+17a] C. Beilschmidt, J. Drönner, M. Mattig, M. Schmidt, C. Authmann, A. Niamir, T. Hickler, and B. Seeger, "Interactive Data Exploration for Geoscience," in *BTW 2017: Datenbanksysteme für Business, Technologie und Web - Workshopband*, vol. P-266, Bonn, Germany: Gesellschaft für Informatik e.V., 2017, pp. 117–126.

[Bei+17b] C. Beilschmidt, J. Drönner, M. Mattig, M. Schmidt, C. Authmann, A. Niamir, T. Hickler, and B. Seeger, "VAT: A Scientific Toolbox for Interactive Geodata Exploration," *Datenbank-Spektrum*, vol. 17, no. 3, pp. 233–243, 2017.

[Bei+17c] C. Beilschmidt, J. Drönner, M. Mattig, and B. Seeger, "VAT: A System for Data-Driven Biodiversity Research," in *EDBT 2017: Proceedings of the 20th International Conference on Extending Database Technology*, Konstanz, Germany: OpenProceedings.org, 2017, pp. 546–549.

[Bei+17d] C. Beilschmidt, T. Fober, M. Mattig, and B. Seeger, "A Linear-Time Algorithm for the Aggregation and Visualization of Big Spatial Point Data," in *SIGSPATIAL '17: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, New York, NY, USA: ACM, 2017, 73:1–73:4.

[Bei+17e] C. Beilschmidt, T. Fober, M. Mattig, and B. Seeger, "Quality Measures for Visual Point Clustering in Geospatial Mapping," in *W2GIS 2017: Proceedings of the 15th International Symposium on Web and Wireless Geographical Information Systems*, Cham, ZG, Switzerland: Springer International Publishing, 2017, pp. 153–168.

[Bei+19] C. Beilschmidt, T. Fober, M. Mattig, and B. Seeger, "An Efficient Aggregation and Overlap Removal Algorithm for Circle Maps," *GeoInformatica*, vol. 23, no. 3, pp. 473–498, 2019.

[Bel11] L. Belbin, "The Atlas of Living Australia's Spatial Portal," in *Proceedings of the Environmental Information Management Conference 2011 (EIM 2011)*, vol. 29, 2011, pp. 39–43.

[BW13] P. Bereuter and R. Weibel, "Real-time Generalization of Point Data in Mobile and Web Mapping Using Quadtrees," *Cartography and Geographic Information Science*, vol. 40, no. 4, pp. 271–281, 2013.

[BS97] A. Berson and S. J. Smith, *Data Warehousing, Data Mining, and OLAP*. New York, NY, USA: McGraw-Hill Education, 1997.

[Bis06] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006, p. 738.

[Bon06] D. G. Bonett, "Confidence Interval for a Coefficient of Quartile Variation," *Computational Statistics and Data Analysis*, vol. 50, no. 11, pp. 2953–2957, 2006.

[BOH11] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 Data-Driven Documents," *IEEE Transactions on Visualization and Computer Graphics*, no. October, pp. 2301–2309, 2011.

[Bro96] M. Brown, "FastCGI: A High-Performance Gateway Interface," in *Programming the Web - A Search for APIs Workshop at Fifth International World Wide Web Conference*, vol. 6, 1996.

[BDF16] P. Buneman, S. Davidson, and J. Frew, "Why Data Citation is a Computational Problem," *Communications of the ACM*, vol. 59, no. 9, pp. 50–57, 2016.

[But+16] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub, "The GeoJSON Format - RFC 7946," Internet Engineering Task Force (IETF), Tech. Rep., 2016.

[Can+07] J. W. Cannon, W. J. Floyd, W. R. Parry, and K. Stephenson, "Introduction to Circle Packing: The Theory of Discrete Analytic Functions," *The Mathematical Intelligencer*, vol. 29, no. 3, pp. 63–66, 2007.

[CMS99] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*. Academic Press, 1999.

[Che+09] M. Chen, D. Ebert, H. Hagen, R. S. Laramee, R. Van Liere, K.-L. Ma, W. Ribarsky, G. Scheuermann, and D. Silver, "Data, Information, and Knowledge in Visualization," *IEEE Computer Graphics and Applications*, vol. 29, no. 1, 2009.

[CMS98] P. Cignoni, C. Montani, and R. Scopigno, "DeWall : A Fast Divide & Conquer Delaunay Triangulation Algorithm in Ed," *Computer-Aided Design*, vol. 5, no. 30, pp. 333–341, 1998.

[CT05] K. A. Cook and J. J. Thomas, *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. Los Alamitos, CA, US: IEEE Computer Society, 2005.

[CF14] G. W. Corder and D. I. Foreman, *Nonparametric Statistics: A Step-by-Step Approach*. Hoboken, NJ, USA: John Wiley & Sons, 2014.

[Cor+09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: The MIT Press, 2009.

[CW14] M. J. Costello and J. Wieczorek, "Best Practice for Biodiversity Data Management and Publication," *Biological Conservation*, vol. 173, pp. 68–73, 2014.

[DA17] P. P. Daniels and L. Atencio, *RxJS in Action*. Greenwich, CT, USA: Manning Publications, 2017.

[Das+12] A. Das Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy, "Efficient Spatial Sampling of Large Geographical Tables Categories and Subject Descriptors," *SIGMOD '12: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 193–204, 2012.

[dBer+08] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin, Heidelberg, Germany: Springer, 2008.

[dBS04] M. de Berg and B. Speckmann, "Computational Geometry," in *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sahni, Eds., Chapman and Hall/CRC, 2004, pp. 62–1–62–20.

[DGK04] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-Means, Spectral Clustering and Normalized Cuts," in *KDD '04: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, New York, NY, USA, 2004, pp. 551–556.

[Di +94] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, "Algorithms for Drawing Graphs: an Annotated Bibliography," *Computational Geometry*, vol. 4, no. 5, pp. 235–282, 1994.

[DGG+14] M. Diepenbroek, F. Glöckner, P. Grobe, *et al.*, "Towards an Integrated Biodiversity and Ecological Research Data Management and Archiving Platform: The German Federation for the Curation of Biological Data (GFBio)," in *GI-Jahrestagung*, 2014, pp. 1711–1721.

[EF13] D. Eddelbuettel and R. François, "Rcpp: Seamless R and C++ Integration," *Journal of Statistical Software*, vol. 40, no. 8, pp. 1–18, 2013.

[ED07] G. Ellis and A. Dix, "A Taxonomy of Clutter Reduction for Information Visualisation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1216–1223, 2007.

[EGA18] N. Escribano, D. Galicia, and A. H. Ariño, "The Tragedy of the Biodiversity Data Commons: A Data Impediment Creeping Nigher?" *The Journal of Biological Databases and Curation*, 2018.

[Est+96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *International Conference on Knowledge Discovery and Data Mining*, vol. 240, 1996, pp. 226–231.

[Fin09] T. Finch, "Incremental Calculation of Weighted Mean and Variance," *General Relativity and Gravitation*, vol. 39, no. 4, pp. 511–520, 2009.

[FCH11] T. Fober, W. Cheng, and E. Hüllermeier, "Focusing Search in Multiobjective Evolutionary Optimization through Preference Learning from User Feedback," in *Proceedings of the 21st Workshop on Computational Intelligence*, vol. 40, Karlsruhe, Germany: KIT Scientific Publishing, 2011, p. 107.

[FC85] D. Forrest and H. W. Castner, "The Design and Perception of Point Symbols for Tourist Maps," *The Cartographic Journal*, vol. 22, no. 1, pp. 11–19, 1985.

[Gam+94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[GT15] J. Gan and Y. Tao, "DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation," in *SIGMOD '15: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 519–530.

[GWU99] H. Garcia-Molina, J. Widom, and J. D. Ullman, *Database System Implementation*. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[Gha+14] T. M. Ghanem, A. Magdy, M. Musleh, S. Ghani, and M. F. Mokbel, "VisCAT: Spatio-Temporal Visualization and Aggregation of Categorical Attributes in Twitter Data," in *SIGSPATIAL '14: Proceedings of the 22th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, New York, NY, USA: ACM, 2014, pp. 537–540.

[GMT07] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering Aggregation," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, pp. 1–30, 2007.

[Góm+13] J. Gómez-Luna, J. M. González-Linares, J. I. Benavides, and N. Guil, "An optimized approach to histogram computation on GPU," *Machine Vision and Applications*, vol. 24, no. 5, pp. 899–908, 2013.

[Gor+17] N. Gorelick, M. Hancher, M. Dixon, S. Ilyushchenko, D. Thau, and R. Moore, "Google Earth Engine: Planetary-scale geospatial analysis for everyone," *Remote Sensing of Environment*, vol. 202, pp. 18–27, 2017.

[Gra13] A. Graser, *Learning QGIS 2.0*. Packt Publishing Ltd, 2013.

[GSH15] T. Gratier, P. Spencer, and E. Hazzard, *OpenLayers 3: Beginner's Guide*. Packt Publishing Ltd, 2015.

[Gri15] R. E. Griffin, "When are Old Data New Data?" *GeoResJ*, vol. 6, pp. 92–97, 2015.

[GS07] C. M. Grinstead and J. L. Snell, *Introduction to Probability*. American Mathematical Society, 2007.

[GB17] M. Gröbe and D. Burghardt, "Micro Diagrams: A Multi-Scale Approach for Mapping Large Categorised Point Datasets," in *Proceedings of AGILE 2017: The 20th AGILE International Conference on Geographic Information Science*, 2017.

[Gru+15] M. G. Gruppi, S. V. G. Magalhães, M. V. A. Andrade, W. R. Franklin, and W. Li, "An Efficient and Topologically Correct Map Generalization Heuristic," *Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS)*, pp. 516–525, 2015.

[Gun13] A. Gunawan, "A Faster Algorithm for DBSCAN," Master's Thesis, Technische University Eindhoven, 2013.

[GBM07] A. Güntsch, W. Berendsohn, and P. Mergen, "The BioCASE Project - A Biological Collections Access Service for Europe," *Ferrantia*, vol. 51, pp. 103–108, 2007.

[Guo+18] T. Guo, K. Feng, G. Cong, and Z. Bao, "Efficient Selection of Geospatial Data on Maps for Interactive and Visualized Exploration," in *SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data*, New York, NY, USA: ACM, 2018, pp. 567–582.

[HGS17] S. Hagedorn, P. Götze, and K.-U. Sattler, "The STARK Framework for Spatio-Temporal Data Analytics on Spark," in *BTW 2017: Datenbanksysteme für Business, Technologie und Web*, vol. P-265, Bonn, Germany: Gesellschaft für Informatik e.V., 2017, pp. 123–142.

[HKP11] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2011.

[HS12] J. Heer and B. Shneiderman, "Interactive Dynamics for Visual Analysis," *Queue*, vol. 10, no. 2, p. 30, 2012.

[HL16] A. Hejlsberg and S. Lucco, *TypeScript Language Specification, Version 1.8*. Microsoft, 2016.

[Hol+13] R. Hollmann, C. J. Merchant, R. Saunders, C. Downy, M. Buchwitz, A. Cazenave, E. Chuvieco, P. Defourny, G. De Leeuw, R. Forsberg, T. Holzer-Popp, F. Paul, S. Sandven, S. Sathyendranath, M. Van Roozendael, and W. Wagner, "The ESA Climate Change Initiative: Satellite Data Records for Essential Climate Variables," *Bulletin of the American Meteorological Society*, vol. 94, no. 10, pp. 1541–1552, 2013.

[HA85] L. Hubert and P. Arabie, "Comparing Partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[HdB09] O. Huisman and R. A. de By, "Principles of Geographic Information Systems," *ITC Educational Textbook Series*, vol. 1, 2009.

[Hül+08] E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker, "Label Ranking by Learning Pairwise Preferences," *Artificial Intelligence*, vol. 172, no. 16-17, pp. 1897–1916, 2008.

[Iac17] S. Iacovella, *GeoServer Beginner's Guide: Share Geospatial Data using Open Source Standards*. Packt Publishing Ltd, 2017.

[JHS13] S. Jänicke, C. Heine, and G. Scheuermann, "GeoTemCo: Comparative Visualization of Geospatial-Temporal Data with Clutter Removal Based on Dynamic Delaunay Triangulations," in *VISIGRAPP 2012: Proceedings of the 7th International Joint Conference on Computer Vision, Imaging and Computer Graphics. Theory and Application*, vol. 359, Berlin, Heidelberg, Germany: Springer, 2013, pp. 160–175.

[Jän+12] S. Jänicke, C. Heine, R. Stockmann, and G. Scheuermann, "Comparative Visualization of Geospatial-Temporal Data," in *GRAPP & IVAPP 2012: Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Application*, Setúbal, Portugal: SciTePress, 2012, pp. 613–625.

[Jas+04] M. W. Jaspers, T. Steen, C. V. D. Bos, and M. Geenen, "The Think Aloud Method: A Guide to User Interface Design," *International Journal of Medical Informatics*, vol. 73, no. 11-12, pp. 781–795, 2004.

[JMG12] W. Jetz, J. M. McPherson, and R. P. Guralnick, "Integrating Biodiversity Distribution Knowledge: Toward a Global Map of Life," *Trends in Ecology and Evolution*, vol. 27, no. 3, pp. 151–159, 2012.

[JWO11] A. C. Jones, R. J. White, and E. R. Orme, "Identifying and relating biological concepts in the Catalogue of Life," *Journal of Biomedical Semantics*, vol. 2, no. 1, p. 7, 2011.

[KB13] A. Kaehler and G. Bradski, *Learning OpenCV - Computer Vision in C++ with the OpenCV Library*. Sebastopol, CA, USA: O'Reilly Media, 2013.

[KS99] K. R. Kanth and A. Singh, "Optimal Dynamic Range Searching in Non-replicating Index Structures," in *Proceedings of the 7th International Conference on Database Theory (ICDT)*, Berlin, Heidelberg, Germany: Springer, 1999, pp. 257–276.

[KML15] T. Karnagel, R. Mueller, and G. M. Lohman, "Optimizing GPU-accelerated Group-By and Aggregation," *ADMS '15: Proceedings of the 6th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, pp. 13–24, 2015.

[KR90] L. Kaufman and P. J. Rousseeuw, "Agglomerative Nesting (Program AGNES)," in *Finding Groups in Data: An Introduction to Cluster Analysis*, L. Kaufman and P. J. Rousseeuw, Eds., Hoboken, NJ, USA: John Wiley & Sons, 1990, pp. 199–252.

[Kei+08a] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler, "Visual Analytics: Scope and Challenges," in *Visual Data Mining*, 4404, S. J. Simoff, M. H. Böhlen, and A. Mazeika, Eds., vol. 4404, Berlin, Heidelberg, Germany: Springer, 2008, pp. 76–90.

[Kei+08b] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual analytics: Definition, Process, and Challenges," in *Information Visualization*, A. Kerren, J. T. Stasko, J.-D. Fekete, and C. North, Eds., Berlin, Heidelberg, Germany: Springer, 2008, pp. 154–175.

[KK94] Z. Kemp and A. Kowalczyk, "Incorporating the Temporal Dimension in a GIS," in *Innovations in GIS*, M. F. Worboys, Ed., vol. 1, London, UK: Taylor and Francis, 1994, pp. 89–102.

[Knu98] D. E. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1998.

[Kon10] A. G. Konheim, *Hashing in Computer Science: Fifty Years of Slicing and Dicing*. Hoboken, NJ, USA: John Wiley & Sons, 2010, p. 386.

[Kör16] M. Körber, "Caching von Geodaten," Unpublished Master's Thesis, University of Marburg, 2016.

[Kra04] M. J. Kraak, "The Rrole of the Map in a Web-GIS environment," *Journal of Geographical Systems*, vol. 6, no. 2, pp. 83–93, 2004.

[KO13] M.-J. Kraak and F. J. Ormeling, *Cartography: Visualization of Spatial Data*. Routledge, 2013.

[Kra67] S. Kravitz, "Packing Cylinders Into Cylindrical Containers," *Mathematics magazine*, vol. 40, no. 2, pp. 65–71, 1967.

[KK13] M. Krstajic and D. A. Keim, "Visualization of Streaming Data: Observing Change and Context in Information Visualization Techniques," *IEEE International Conference on Big Data*, pp. 41–47, 2013.

[LH11] O. D. Lampe and H. Hauser, "Interactive Visualization of Streaming Data with Kernel Density Estimation," in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, IEEE, 2011, pp. 171–178.

[LPB16] H. U. Leena, B. G. Premasudha, and P. K. Basavaraja, "Sensible Approach for Soil Fertility Management using GIS Cloud," in *ICACCI '16: 2016 International Conference on Advances in Computing, Communications and Informatics*, Washington, DC, USA: IEEE Computer Society, 2016, pp. 2776–2781.

[Li+05] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "Semantics and Evaluation Techniques for Window Aggregates in Data Streams," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 2005, pp. 311–322.

[LL12] A. Lipowski and D. Lipowska, "Roulette-Wheel Selection via Stochastic Acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.

[LH14] Z. Liu and J. Heer, "The Effects of Interactive Latency on Exploratory Visual Analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2122–2131, 2014.

[LJH13] Z. Liu, B. Jiang, and J. Heer, "imMens: Real-time Visual Querying of Big Data," *Computer Graphics Forum*, vol. 32, no. 3, pp. 421–430, 2013.

[Llo82] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[Löf+17] F. Löffler, K. Opasjumruskit, N. Karam, D. Fichtmüller, U. Schindler, F. Klan, C. Müller-Birn, and M. Diepenbroek, "Honey Bee Versus Apis Mellifera: A Semantic Search for Biological Data," in *The Semantic Web: ESWC 2017 Satellite Events*, Cham, Switzerland: Springer International Publishing, 2017, pp. 98–103.

[Lon+05] P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind, *Geographic Information Systems and Science*. Hoboken, NJ, USA: John Wiley & Sons, 2005.

[LRH07] R. H. Lopes, I. Reid, and P. R. Hobson, "The Two-Dimensional Kolmogorov-Smirnov Test," *Proceedings of Science*, 2007.

[Mac04] A. M. MacEachren, *How Maps Work: Representation, Visualization, and Design*. Guilford Press, 2004.

[Mac11] R. Maciejewski, "Data Representations, Transformations, and Statistics for Visual Reasoning," *Synthesis Lectures on Visualization*, vol. 2, no. 1, pp. 1–85, 2011.

[Mad+12] J. Madhavan, S. Balakrishnan, K. Brisbin, H. Gonzalez, N. Gupta, A. Y. Halevy, K. Jacqmin-Adams, H. Lam, A. Langen, and H. Lee, "Big Data Storytelling Through Interactive Maps," *IEEE Data Engineering Bulletin*, vol. 35, no. 2, pp. 46–54, 2012.

[MNV12] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The Planar k-Means Problem is NP-hard," *Theoretical Computer Science*, vol. 442, pp. 13–21, 2012.

[MAS16] S. T. Mai, I. Assent, and M. Storgaard, "AnyDBC: An Efficient Anytime Density-based Clustering Algorithm for Very Large Complex Datasets," in *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, 2016, pp. 1025–1034.

[Mal13] D. H. Maling, *Coordinate Systems and Map Projections*. Elsevier, 2013.

[Mar11] C. Marin, "WebGL Specification," *Khronos WebGL Working Group*, 2011.

[McL+13] M. McLennan, S. Clark, F. McKenna, E. Deelman, M. Rynge, K. Vahi, D. Kearney, and C. Song, "Bringing Scientific Workflow to the Masses via Pegasus and HUBzero," in *IWSG 2013: Proceedings of the 5th International Workshop on Science Gateways*, 2013.

[Men03] J. Mena, "State of the Art on Automatic Road Extraction for GIS Update: A Novel Classification," *Pattern Recognition Letters*, vol. 24, no. 16, pp. 3037–3058, 2003.

[MR05] L. Meng and T. Reichenbacher, "Map-based Mobile Services: Theories, Methods and Implementations," in *Human Genetics*, L. Meng, T. Reichenbacher, and A. Zipf, Eds., Berlin, Heidelberg, Germany: Springer, 2005, pp. 1–10.

[Mos17] T. Mostak, "Using GPUs to Accelerate Data Discovery and Visual Analytics," in *FTC 2016: Proceedings of Future Technologies Conference*, Washington, DC, USA: IEEE Computer Society, 2017, pp. 1310–1313.

[MP80] J. I. Munro and M. S. Paterson, "Selection and Sorting with Limited Storage," *Theoretical Computer Science*, vol. 12, no. 3, pp. 315–323, 1980.

[Mun14] T. Munzner, *Visualization Analysis and Design*. Boca Raton, FL, USA: CRC Press, 2014.

[ML14] F. Murtagh and P. Legendre, "Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?" *Journal of Classification*, vol. 31, no. 3, pp. 274–295, 2014.

[OH15] R. O. Obe and L. S. Hsu, *PostGIS in Action*, 2nd ed. Greenwich, CT, USA: Manning Publications, 2015.

[Ope06] Open Geospatial Consortium, "Web Map Server (WMS) Implementation Specification," *OpenGIS Project Document*, 2006.

[Ope07] Open Geospatial Consortium, "Web Service Common Implementation Specification," *OpenGIS Project Document*, 2007.

[Ope08] Open Geospatial Consortium, "Web Coverage Service (WCS) Implementation Standard," *OpenGIS Project Document*, 2008.

[Ope10a] Open Geospatial Consortium, "OpenGIS Implementation Standard for Geographic Information - Simple Feature Access," *OpenGIS Project Document*, 2010.

[Ope10b] Open Geospatial Consortium, "OpenGIS Web Feature Service (WFS) 2.0 Interface Standard," *OpenGIS Project Document*, 2010.

[Pan+18] V. Pandey, A. Kipf, T. Neumann, and A. Kemper, "How Good are Modern Spatial Analytics Systems?" *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1661–1673, 2018.

[Pap+05] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 41–82, 2005.

[PCM16] Y. Park, M. J. Cafarella, and B. Mozafari, "Visualization-Aware Sampling for Very Large Databases," in *Proceedings of the IEEE 32nd International Conference on Data Engineering (ICDE)*, Washington, DC, USA: IEEE Computer Society, 2016, pp. 755–766.

[Pas02] N. Paskin, "Digital Object Identifiers," *Information Services and Use*, vol. 22, no. 2-3, pp. 97–112, 2002.

[PD96] J. M. Patel and D. J. DeWitt, "Partition Based Spatial-Merge Join," in *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 1996, pp. 259–270.

[Péb08] P. Pébay, "Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments," Sandia National Laboratories, Tech. Rep., 2008.

[Peb17] E. Pebesma, "The GRASS GIS Temporal Framework: Object oriented code design with examples," *International Journal of Geographical Information Science*, vol. 31, no. 2014, pp. 1–19, 2017.

[PB05] E. Pebesma and R. S. Bivand, "Classes and Methods for Spatial Data: the sp Package," *R News*, vol. 5, no. 2, pp. 9–13, 2005.

[Per+13] H. M. Pereira, S. Ferrier, M. Walters, G. N. Geller, R. H. Jongman, R. J. Scholes, M. W. Bruford, N. Brummitt, S. H. Butchart, A. C. Cardoso, N. C. Coops, E. Dulloo, D. P. Faith, J. Freyhof, R. D. Gregory, C. Heip, R. Höft, G. Hurtt, W. Jetz, D. S. Karp, M. A. McGeoch, D. Obura, Y. Onoda, N. Pettorelli, B. Reyers, R. Sayre, J. P. Scharlemann, S. N. Stuart, E. Turak, M. Walpole, and M. Wegmann, "Essential Biodiversity Variables," *Science*, vol. 339, no. 6117, pp. 277–278, 2013.

[Pet15] M. P. Peterson, "Evaluating Mapping APIs," in *Modern Trends in Cartography*, J. Brus, A. Vondrakova, and V. Vozenilek, Eds., Cham, Switzerland: Springer International Publishing, 2015, pp. 183–197.

[Pic17] S. Pickering, "A New Way to Proxy Levels of Infrastructure Development," *Research and Politics*, vol. 4, no. 1, 2017.

[Pin88] J. Pineda, "A Parallel Algorithm for Polygon Rasterization," in *SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer graphics and Interactive Techniques*, ACM, New York, NY, USA, 1988, pp. 17–20.

[Pow11] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[Rap+14] G. Rapacciuolo, D. B. Roy, S. Gillings, and A. Purvis, "Temporal Validation Plots: Quantifying how well correlative species distribution models predict species' range changes over time," *Methods in Ecology and Evolution*, vol. 5, no. 5, pp. 407–420, 2014.

[Rau+16] A. Rauber, A. Asmi, D. Van Uytvanck, and S. Pröll, "Identification of Reproducible Subsets for Data Citation, Sharing and Re-Use," *Bulletin of IEEE Technical Committee on Digital Libraries, Special Issue on Data Citation*, vol. 12, no. 1, pp. 6–15, 2016.

[RD90] R. Rew and G. Davis, "NetCDF: An Interface for Scientific Data Access," *IEEE Computer Graphics and Applications*, vol. 10, no. 4, pp. 76–82, 1990.

[RR97] N. Ritter and M. Ruth, "The GeoTiff Data Interchange Standard for Raster Geographic Images," *International Journal of Remote Sensing*, vol. 18, no. 7, pp. 1637–1647, 1997.

[Rot13] R. E. Roth, "Interactive Maps: What we know and what we need to know," *Journal of Spatial Information Science*, vol. 6, no. 6, pp. 59–115, 2013.

[Sam84] H. Samet, "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, vol. 16, no. 2, pp. 187–260, 1984.

[Sam04] H. Samet, "Multidimensional Spatial Data Structures," in *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sahni, Eds., Chapman and Hall/CRC, 2004, pp. 16–1–16–29.

[Sam06] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Francisco, CA, USA: Morgan Kaufmann, 2006.

[Sar+12] A. D. Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy, "Efficient Spatial Sampling of Large Geographical Tables Categories and Subject Descriptors," in *SIGMOD '12: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 2012, pp. 193–204.

[SC17] V. Savkin and J. Cross, *Essential Angular*. Packt Publishing, 2017.

[Sco15] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*. Hoboken, NJ, USA: John Wiley & Sons, 2015.

[SW04] B. Seeger and P. Widmayer, "Geographic Information Systems," in *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sahni, Eds., Chapman and Hall/CRC, 2004, pp. 55–1–55–22.

[Shn96] B. Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," in *Proceedings of the 1996 IEEE Symposium on Visual Languages*, Washington, DC, USA: IEEE Computer Society, 1996, pp. 336–343.

[SvW08] Y. B. Shrinivasan and J. J. van Wijk, "Supporting the Analytical Reasoning Process In Information Visualization," *CHI '08: Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems*, pp. 1237–1246, 2008.

[Sib73] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 1973.

[Sin16] D. A. Sinclair, "S-Hull : A Fast Radial Sweep-Hull Routine for Delaunay Triangulation," *CoRR*, pp. 1–7, 2016.

[SM98] J. A. Slater and S. Malys, "WGS 84 — Past, Present and Future," in *Advances in Positioning and Reference Frames*, F. K. Brunner, Ed., Berlin, Heidelberg, Germany: Springer, 1998, pp. 1–7.

[Slo+09] T. Slocum, R. McMaster, F. Kessler, and H. Howard, *Thematic Cartography and Geovisualization*. Upper Saddle River, NJ, USA: Prentice Hall, 2009.

[Ste+13] C. A. Steed, D. M. Ricciuto, G. Shipman, B. Smith, P. E. Thornton, D. Wang, X. Shi, and D. N. Williams, "Big Data Visual Analytics for Exploratory Earth System Simulation Analysis," *Computers & Geosciences*, vol. 61, pp. 71–82, 2013.

[SGS10] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[Stu26] H. A. Sturges, "The Choice of a Class Interval," *Journal of the American Statistical Association*, vol. 21, no. 153, pp. 65–66, 1926.

[The10] The SciDB Development Team, "Overview of SciDB - Large Scale Array Storage, Processing and Analysis," in *SIGMOD '10: ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 2010, pp. 963–968.

[TH07] J. Ticheler and J. U. Hielkema, "GeoNetwork opensource: Internationally Standardized Distributed Spatial Information Management," *OSGeo Journal*, vol. 2, no. 1, pp. 1–6, 2007.

[VCH10] L. Vendramin, R. J. G. B. Campello, and E. R. Hruschka, "Relative Clustering Validity Criteria: A Comparative Overview," *Statistical Analysis and Data Mining*, vol. 3, no. 4, pp. 209–235, 2010.

[Vit85] J. S. Vitter, "Random Sampling with a Reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[Wan+12] J.-f. Wang, A. Stein, B.-b. Gao, and Y. Ge, "A Review of Spatial Sampling," *Spatial Statistics*, vol. 2, pp. 1–14, 2012.

[Wan+15] L. Wang, R. Christensen, F. Li, and K. Yi, "Spatial Online Sampling and Aggregation," *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 84–95, 2015.

[WYM97] W. Wang, J. Yang, and R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," in *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, San Francisco, CA, USA: Morgan Kaufmann Publishers, 1997, pp. 186–195.

[War08] F. Warmerdam, "The Geospatial Data Abstraction Library," in *Open Source Approaches in Spatial Data Handling SE - Advances in Geographic Information Science*, B. Hall and M. G. Leahy, Eds., vol. 2, Springer, 2008, pp. 87–104.

[Wel62] B. P. Welford, "Note on a Method for Calculating Correct Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[Wie+12] J. Wieczorek, D. Bloom, R. Guralnick, S. Blum, M. Döring, R. Giovanni, T. Robertson, and D. Vieglais, "Darwin Core - An Evolving Community-Developed Biodiversity Data Standard," *PLOS ONE*, vol. 7, no. 1, pp. 1–8, 2012.

[WF09] L. Wilkinson and M. Friendly, "The History of the Cluster Heat Map," *The American Statistician*, vol. 63, no. 2, pp. 179–184, 2009.

[Win13] J. Winn, "Research Data Management using CKAN: A Datastore, Data Repository and Data Catalogue," in *IASSIST 2013: Data Innovation: Increasing Accessibility, Visibility,and Sustainability*, 2013.

[Win18] K. Winter, "Visual Clustering of Large Heterogeneous Point Sets," Unpublished Bachelor's Thesis, University of Marburg, 2018.

[WMB99] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[Wol+13] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna Workflow Suite: Designing and Executing Workflows of Web Services on the Desktop, Web or in the Cloud," *Nucleic Acids Research*, vol. 41, no. Web Server issue, W557–W561, 2013.

[YWS15] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: A Cluster Computing Framework for Processing Large-scale Spatial Data," in *SIGSPATIAL '15: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, New York, NY, USA: ACM, 2015, 70:1–70:4.

[Zah+10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *HotCloud '10: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[Zah70] C. T. Zahn, "Graph Theoretical Methods for Detecting and Describing Gestalt Clusters," *IEEE Transactions on Computers*, vol. C-20, no. 1, pp. 68–86, 1970.

[Zas15] M. Zastrow, "Data Visualization: Science on the Map," *Nature (Toolbox)*, vol. 519, no. 7541, pp. 119–120, 2015.

[ZYG14] J. Zhang, S. You, and L. Gruenwald, "Data Parallel Quadtree Indexing and Spatial Query Processing of Complex Polygon Data on GPUs," in *ADMS '14: Proceedings of the 5th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 2014, pp. 13–24.

[ZYG17] J. Zhang, S. You, and L. Gruenwald, "Towards GPU-Accelerated Web-GIS for Query-Driven Visual Exploration," in *W2GIS 2017: Proceedings of the 15th International Symposium on Web and Wireless Geographical Information Systems*, Cham, ZG, Switzerland: Springer International Publishing, 2017, pp. 119–136.

[Zha+12] L. S. Zhang, A. Stoffel, M. Behrisch, S. Mittelstadt, T. Schreck, R. Pompl, S. Weber, H. Last, and D. Keim, "Visual Analytics for the Big Data Era - A Comparative Review of State-of-the-Art Commercial Systems," in *VAST '12: Proceedings of the 2012 IEEE Conference on Visual Analytics Science and Technology*, Washington, DC, USA: IEEE Computer Society, 2012, pp. 173–182.

[Zha+16a] L. Zhang, C. Rooney, L. Nachmanson, B. L. W. Wong, B. C. Kwon, F. Stoffel, M. Hund, N. Qazi, U. Singh, and D. A. Keim, "Spherical Similarity Explorer for Comparative Case Analysis," in *Proceedings of the IS&T International Symposium on Electronic Imaging 2016: Visualization and Data Analysis*, Oxford, UK: Ingenta, 2016, pp. 1–10.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," in *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, 1996, pp. 103–114.

[Zha+16b] Z. Zhang, J. Li, X. Li, Y. Lin, S. Zhang, and C. Wang, "A Fast Method for Measuring the Similarity Between 3D Model and 3D Point Cloud," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 41, pp. 725–728, 2016.

List of Figures

List of Tables

List of Algorithms

Curriculum Vitae

This page contains personal data. It is therefore not part of the online publication.