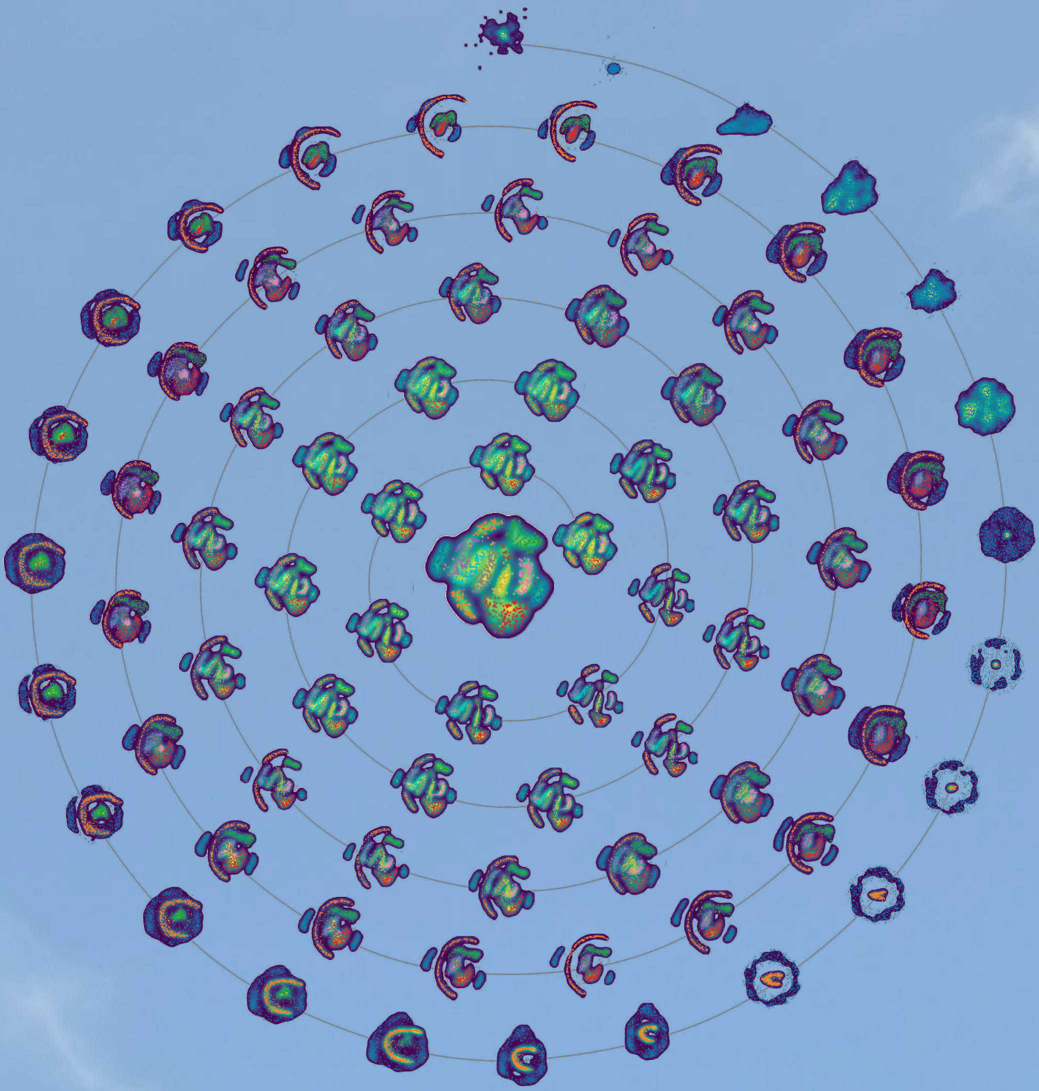
Progressive Data Analysis

Roadmap and Research Agenda

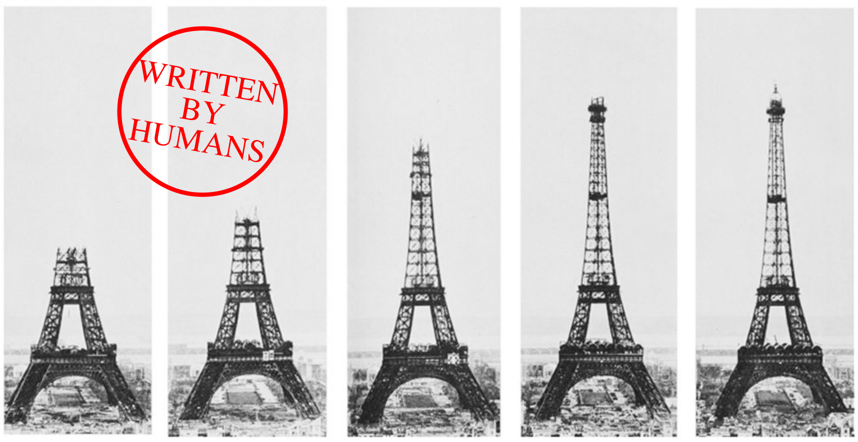


Editors: Jean-Daniel Fekete, Danyel Fisher and Michael Sedlmair

Progressive Data Analysis

Roadmap and Research Agenda

Editors: Jean-Daniel Fekete,
Danyel Fisher, and
Michael Sedlmair



Published by the Eurographics Association
—Groene Loper 3, 5612AE Eindhoven, The Netherlands.

Printed by BELTZ Grafische Betriebe GmbH, Bad Langensalza, Germany.

Cover image © by Véronique Fekete

ISBN: 978-3-03868-270-7

The electronic version of this work is available from the Eurographics Digital Library:
`https://doi.org/10.2312/pda.20242707`

Preface

We live in an era in which data is abundant and growing rapidly. Big data databases sprawl past memory and computation limits and across distributed systems. To sustain this growth, engineers are designing new hardware and software systems with new storage management and capabilities for predictive computation. Yet, as datasets grow and computations become more complex, response times suffer. These infrastructures, while good for data at scale, do not support exploratory data analysis (EDA) effectively. EDA allows analysts to make sense of data with little or no known model and is essential in many application domains, from network security and fraud detection to epidemiology and preventive medicine. Data exploration is conducted with an iterative loop in which analysts interact with data through computations that return results, usually displayed with visualizations, which the analyst can then interact with again. EDA requires highly responsive system response times: at 500 ms, users typically change their querying behavior; after five or ten seconds, they abandon tasks or lose attention. To address this problem, a new computational paradigm has emerged in the last decade, which goes under several names. In the database community, it is called *online aggregation*, while among visualization researchers, it has been called *progressive*, *incremental*, or *iterative visualization*. In this book, we will refer to it as *Progressive Data Analysis*.

This paradigm consists of splitting long computations into a series of approximate results that improve over time. In this process, partial or approximate results are rapidly returned to the user and can be interacted with in a fluent and iterative fashion. With the increasing growth in data, such progressive data analysis approaches will become one of the leading paradigms for data exploration systems, but it will also require major changes to algorithms, data structures, and visualization tools.

By solving the latency issue, progressive data analysis opens up new perspectives but also presents new challenges. Problems involving complex analyses can now be addressed interactively with little or no preparation time, but how long should the analyst wait before obtaining results that are good enough to make decisions? Can progressive systems provide effective quality measures to avoid either waiting too long or deciding too early? While the progressive process is being computed, the iterative visualization of partial results should remain stable enough to be monitored.

Can we stabilize these partial results without hurting their quality or speed? And, more fundamentally, can we transform all data analysis operations so that they become progressive? These questions, among others, are at the core of this book.

This book is an introduction to the new paradigm of progressive data analysis and visualization. It explores the major scientific and technical benefits of performing complex data analysis progressively on big data. It also examines the challenges that must be addressed for the paradigm to become fully usable. These important issues involve research fields that are traditionally viewed as separate areas in computer science: databases, scientific computing, machine learning, visualization, statistics, and human-computer interaction; they will need to work together to achieve end-to-end solutions to solve these challenges and enable the emergence of practical, progressive systems. This book closes with a research agenda to help researchers converge on key questions.

Acknowledgments

This book is the result of the Dagstuhl seminar 18411 “Progressive Data Analysis and Visualization”. We want to thank Schloss Dagstuhl—Leibniz Center for Informatics for its support and the outstanding facility it provides to researchers. We also thank all the participants for the fruitful discussions.

Many people helped prepare this book, in addition to the chapter authors and editors. It was written mostly during the COVID-19 crisis while several authors struggled to manage their health, professional, and family life. We are grateful for their scientific dedication.

We want to thank: Anastasia Bezerianos, Remco Chang, Min Chen, Pierre Dragicevic, Steven Drucker, Nivan Roberto Ferreira, Dominik Moritz, Jaakko Peltonen, Adam Perer, Christian Poli, Marianne Procopio, Guido Reina, Daniel Weiskopf, Eugene Wu, and anonymous reviewers for proofreading chapters of this book and providing insightful feedback.

Jean-Daniel Fekete, Danyel Fisher, Michael Sedlmair, Editors
Paris, Seattle, Stuttgart, November 10, 2024

Contents

Preface **i**

1 Introduction **1**
1.1 Intuition . . . 2
1.2 Why we need Progressive Data Analysis . . . 4
1.3 Illustrative Scenario . . . 6
1.4 Resistance to Progressive Data Analysis? . . . 9
1.5 A History of Progressive Data Analysis . . . 10
1.6 Goal of the Book . . . 10
1.7 A Research Future for PDA . . . 12
1.8 Organization of this Book . . . 13

2 Concepts and Definition **16**
2.1 Motivation . . . 16
2.2 Concepts . . . 18
2.3 Definitions . . . 22
2.4 *Progressive-in-chunks* and *Progressive-in-iterations* . . . 30
2.5 Summary . . . 31

3 Data Management **33**
3.1 Motivation . . . 33
3.2 Requirements for Progressive Data Management . . . 36
3.3 Computing Progressive Results . . . 37
3.4 Progressive Queries . . . 39
3.5 Progressive Data Exploration Systems . . . 42
3.5.1 OLA-RAW . . . 42
3.5.2 ProgressiveDB . . . 45
3.6 Extended Services for Progressive Data Management . . . 47
3.7 Summary . . . 48

4 Data Structures and Algorithms **49**
4.1 Vision . . . 51

4.2 Motivation 51
4.2.1 Streaming and Steerable Visualization 52
4.2.2 Creating a Progressive Algorithm 53
4.2.3 Uncertainty on Aggregated Results 54
4.3 Basic Data Structures for Progressivity 54
4.3.1 Vectors, Multidimensional Arrays, and Tensors 55
4.3.2 Data Tables 56
4.3.3 Networks 56
4.4 Basic Algorithms 57
4.4.1 Broadcasting Operations 58
4.4.2 Linear Algebra 59
4.4.3 Statistical Operations 60
4.4.4 Data Sketching 60
4.5 Input and Output 61
4.5.1 Detailed Input 61
4.6 Specific Progressive Services and Strategies 64
4.6.1 Progressive Stability and Normalization 64
4.6.2 Data Change Management 65
4.7 Summary 67

5 Visualization **69**
5.1 From Classic to Scalable Visualization 70
5.2 From Scalable to Progressive Visualization 73
5.2.1 Progressive Rendering 73
5.2.2 Interaction, Steering, and Control 78
5.3 Designing a Progressive Visual Analytics System 79
5.3.1 Choose the Right Visual Mapping 79
5.3.2 Represent the Changes 80
5.3.3 Visualizing the Progression 81
5.3.4 Interaction Between Visualization and Progression 82
5.4 Existing Visualization Approaches in PDA 82
5.4.1 DimXplorer 83
5.4.2 InsightsFeed 85
5.4.3 ProReveal 87
5.5 Challenges for Progressive Visual Analytics 89
5.6 Summary 90

6 Uncertainty and Quality **92**
6.1 Introduction . . . 92
6.2 Example Pipelines . . . 95
6.3 Sources of *PDA Uncertainty* . . . 96
6.3.1 Uncertainty from Data Processing . . . 96
6.3.2 Uncertainty from Computation . . . 97
6.3.3 Uncertainty from the Progression . . . 98
6.4 Measures/Quantification of Uncertainty . . . 99
6.4.1 Sampling Approximations . . . 100
6.4.2 Aggregation Approximates . . . 103
6.4.3 Computation-based Approximates . . . 104
6.4.4 Progressive Measures . . . 105
6.5 Visualization of *PDA Uncertainty* . . . 105
6.6 Summary . . . 107

7 Human Aspects **108**
7.1 Time Engineering for Human Factors . . . 110
7.2 User Roles . . . 111
7.2.1 Role 1: The Progressive Observer . . . 111
7.2.2 Role 2: The Progressive Searcher . . . 112
7.2.3 Role 3: The Progressive Explorer . . . 113
7.3 User Tasks . . . 114
7.3.1 The Observer's Analytic Tasks . . . 115
7.3.2 The Searcher's Analytic Tasks . . . 116
7.3.3 The Explorer's Analytic Tasks . . . 116
7.4 User Focus . . . 117
7.4.1 F1: Focus on the Data Space . . . 118
7.4.2 F2: Focus on the Algorithm Space . . . 119
7.5 User Complications and Biases . . . 120
7.6 Application Example: Crisis Management . . . 124
7.6.1 The Background . . . 124
7.6.2 An Illustrating Scenario . . . 126
7.6.3 The Design Guidance . . . 128
7.7 Summary . . . 130

8 Progressive Methods in Machine Learning **132**
8.1 Introduction . . . 132

8.2 Motivation 134
8.3 Progressive Machine Learning Algorithms 135
8.3.1 Characteristics of ML Algorithms 135
8.4 Examples of Progressive ML Algorithms 138
8.4.1 *k*-Nearest Neighbors Classifier 138
8.4.2 Support Vector Machines as Classifiers 140
8.4.3 Deep Neural Networks 143
8.5 Challenges and Recent Advances 145
8.6 Summary 147

9 Evaluation **149**
9.1 Scenario and Challenges 150
9.1.1 Approaches to Evaluation 150
9.1.2 Performance Measures 152
9.2 Evaluation at Multiple Stages of the Pipeline 153
9.2.1 Data Management 153
9.2.2 Algorithms 156
9.2.3 Visualization and Interaction 158
9.2.4 Human Factors: Perception and Cognition 163
9.2.5 Machine Learning 166
9.2.6 System Level 167
9.3 Summary 169

10 Challenges and Research Agenda **171**
10.1 Overarching Challenges to Broader Adoption of PDA 171
10.2 Data Management 172
10.3 Data Structures and Algorithms 173
10.4 Visualization 174
10.5 Quality and Uncertainty 176
10.6 Human Aspects 177
10.7 Machine Learning 177
10.8 Evaluation 178
10.9 Threats and Opportunities 180

11 Conclusion **182**

Bibliography **184**

List of Figures **209**

List of Tables **211**

List of Definitions **212**

Listings **213**

Alphabetical Index **215**

List of Authors

Chapter 1	Jean-Daniel Fekete	Inria & Université Paris-Saclay
	Danyel Fisher	
	Michael Sedlmair	University of Stuttgart
Chapter 2	Michaël Aupetit	Qatar Computing Research Institute
	Giuseppe Santucci	Sapienza, Università di Roma
Chapter 3	Florin Rusu	University of California Merced
	Chris Weaver	University of Oklahoma
	Carsten Binnig	Technical University of Darmstadt & DFKI
Chapter 4	Jean-Daniel Fekete	Inria & Université Paris-Saclay
Chapter 5	Marco Angelini	Sapienza, Università di Roma
	Jaemin Jo	Sungkyunkwan University
Chapter 6	Anna Vilanova	Eindhoven University of Technology
	Marco Angelini	Sapienza, Università di Roma
	Sriram Karthik Badam	Apple Inc.
	Jean-Daniel Fekete	Inria & Université Paris-Saclay
Chapter 7	Hans-Jörg Schulz	Aarhus University
	Michaël Aupetit	Qatar Computing Research Institute
	Danyel Fisher	
Chapter 8	Cagatay Turkay	University of Warwick
	Nicola Pezzotti	Philips Cardiologs & Eindhoven University of Technology & AI4MRI LUMC
	Hendrik Strobelt	IBM Research, MIT-IBM Watson AI Lab
	Barbara Hammer	CITEC, Bielefeld University
Chapter 9	Gaëlle Richer	Inria & Université Paris-Saclay
	Jean-Daniel Fekete	Inria & Université Paris-Saclay
	Michael Sedlmair	University of Stuttgart
Chapter 10 & 11	Jean-Daniel Fekete	Inria & Université Paris-Saclay
	Danyel Fisher	
	Michael Sedlmair	University of Stuttgart

1 Introduction

1.1 Intuition 2
1.2 Why we need Progressive Data Analysis 4
1.3 Illustrative Scenario 6
1.4 Resistance to Progressive Data Analysis? 9
1.5 A History of Progressive Data Analysis 10
1.6 Goal of the Book 10
1.7 A Research Future for PDA 12
1.8 Organization of this Book 13

Authors Jean-Daniel Fekete, Danyel Fisher, and Michael Sedlmair

Books on data always start with the usual clichés about the wealth of data being generated nowadays, and this one is no exception. What we are looking for is a way to allow the exploration of data at scale, but the fact is that there is currently no satisfactory solution that would allow an analyst to dive into a large dataset and make sense of it using current data exploration methods. The scale of existing exploratory systems is limited for reasons that will not be resolved by improving processor speed and network bandwidth. We are reaching physical limits for computer and data communication speed.

As datasets grow and computations become more complex, the response time of interactive systems suffers. The progressive paradigm has emerged over the last decade to address this problem. The Progressive Data Analysis (PDA) paradigm splits long computations into a series of approximate results that improve with time. For example, by looking at smaller subsets of the data, the user quickly sees a result and can decide how to respond to it—whether to move to the next question or wait for further refinement of the result. PDA offers data analysts a trade-off between time lost while waiting and accuracy of results. Choosing the shortest amount of time to make a decision in order to maximize an expected reward is an instance of the well-known "optimal stopping" problem [52].

In this book, we will describe PDA as a human-in-the-loop process that supports data exploration at scale. Humans input queries to a system that would ordinarily have a long response time either because of long computation times or large data operations. The user accepts less accuracy of responses in exchange for low latency. PDA is “progressive” in that we expect to receive more precise responses over time, either by examining increasingly large samples or by continuing a slow computation.

Coupled with interactive visualization, PDA becomes Progressive Visual Analytics (PVA), which allows analysts to conduct data exploration at scale and to interact with complex computations such as statistical analyses, machine learning, and simulations.

PDA and PVA are both based on the principle of continuous control, in which analysts are able to interact with the computer during computation, for example, by adjusting computation parameters based on actual results, a process called *steering*.

Systems that use the PDA paradigm are designed to take into account human capabilities (e.g., monitoring, controlling, or steering the computation as it progresses), as well as limitations, particularly regarding latency and attention. As we know from research in Human-Computer Interaction (HCI) [179, 199, 281], humans who interact with computers need to receive a response from the system with bounded latency to avoid dramatic declines in efficiency. While PDA can alleviate the latency issue to some extent, as Zgraggen *et al.* [342] have shown, progressivity introduces several other complications for computer systems and human cognition that must be addressed. This book is an attempt to identify these issues and challenges and to lay out a research agenda to address them.

As we will see in the following chapter, PDA decouples the latency issue from computational complexity, but at the cost of having to assess and monitor the quality of the results.

PDA can also be seen as a broad net approach: a system might provide regular updates—or it might provide a first approximation and only produce high-quality results on demand. These different approaches allow PDA systems to be tailored to their usage patterns and data environments.

1.1 Intuition

We are already familiar with progressive processes from everyday life; the computer-supported examples are fundamentally no different. In this section, we use an example of a United States presidential election to help develop an intuition of how Progressive Data Analysis works.

In the last few election cycles in the United States, the world has seen the challenges of Progressive Data Analysis in action as it struggled through the aftermath of the US presidential election. Although by election day, every voter had decided and every vote had been cast, the process of receiving, validating, and counting those votes would be spread out over weeks.

The process of counting ballots in a national US election is an example of Progressive Data Analysis. Two candidates compete for the largest number of electoral votes. Computationally, the process can be modeled in three simple phases:

1. The tallying of the number of votes for each candidate in each electoral district,
2. The assignment of electoral votes based on these numbers,
3. The selection of the candidate with the largest number of electoral votes, while checking for any possible fraud and error.

Each phase can be computed precisely, and accurate results can be expected to arrive after a certain amount of time. This delay, however, is unsatisfactory for many stakeholders: for the news media that wants to announce the results; for the candidates who want to know if they won.

Looking at the results progressively can help us predict the results of the contest and allow candidates to begin transitioning to their new jobs. As batches of ballots arrive, there are several tasks that are specific to PDA:

1. An estimation of the likelihood that the outcome will be in favor of one candidate vs. the other based on intermediate results,
2. Detecting and addressing possible fraud or errors as early in the process as possible,
3. Identification of what future information—such as critical county results—might improve the quality of estimates most and the elimination of results that will have no effect on the outcome.

As we saw during the post-election process, anyone attempting to figure out who the next president was going to be encountered many challenges. How good were the current results as a predictor of the outcome? Were the votes that had been tallied thus far a representative sample of what would be counted later? Journalists struggled to find the right language to describe what was happening, making remarks about whether one candidate had "momentum" or "a lead"—language that implied that new votes were somehow still being cast or new information was emerging.

Even for a national election in an experienced country, monitoring and controlling this seemingly simple progressive process can be challenging. Would the process run more smoothly if the final decision were not made public until the end? If the process were flawless, monitoring would just be a source of distraction for some and entertainment for others. However, where the process has potential issues, the failure to monitor it progressively would raise suspicions and create problems.

In the best cases, we can expect to see a clear trend early in the process and we can be confident about the outcome early on without having to closely monitor the process. These early decisions are essential in optimizing the time we spend on time-consuming processes and in optimizing energy consumption by stopping any costly computations early, an issue that is relevant in green computing, a topic that has been increasingly receiving attention.

This book will highlight *when* PDA might be useful or necessary, *what* its main benefits, issues, and open questions are, and *how* it should be used and implemented. We can already say with certainty that one of PDA's most exciting aspects is the way that it opens up so many opportunities for the exploration of data that were previously impossible. However, PDA systems pose implementation challenges because they use a paradigm that differs in important ways from those of traditional programming and execution semantics. Therefore, it will require effort and time to design and implement mature, progressive systems, and many parts of the traditional data analysis software "stack" will need adaptation.

1.2 Why we need Progressive Data Analysis

The ability to explore data interactively remains as important as ever, maybe even more so given the importance of data today. But unfortunately, exploration is still lagging behind. We constantly hear optimistic claims about how we are now able to obtain perfect data, processed by perfect systems providing perfect results using a well-known objective function. The result of this has been to obscure the importance of exploration. No research domain or industry wants to advertise the issues they're confronting with data. Yet, the reality is something quite different. Data is rarely clean, analysis systems are rarely perfect—sometimes far from it—and it often delivers results that at the very least need to be verified (if not discarded outright) and require humans to assess its utility. But how would we know any of this without exploration tools?

Initially, Exploratory Data Analysis (EDA) [303] was designed to help statisticians make sense of data, which assumed that the data was already available and small enough in size to be visualized using the basic visual representations: bar charts, line charts, histograms, scatterplots, and a few others. These assumptions have changed in important ways. Today, the statistician has been replaced by a range of different analysts working to make sense of data—including, in addition to engineers and scientists, ordinary citizens, activists, journalists, humanists, and marketing specialists.

Contemporary data exploration has become an extremely diverse field, encompassing data scrutiny, data cleaning, and more generally, data wrangling [150]. It also includes the exploration of statistical data, website and program logs, simulation results, and models fitted by machine learning, often applied to dynamic data.

The exploration of server logs can reveal intrusions, malfunctions, and interesting navigation patterns. Cars, trains, and planes now collect telemetry data that can reveal important information about intermittent problems, worn parts, or even traffic patterns. By simulating power production in a country, for example, thousands of possible scenarios can be generated allowing us to predict a vast range of possible futures, some of which will be impossible, while others are unexpected. Through exploration we are able to discover these among the myriad of well-understood scenarios, as well as to uncover possible bugs in the simulators. AutoML [112] optimizes machine learning pipelines by trying many possible pipelines with even larger numbers of hyper-parameters. In reality, comparing the results of these pipelines requires exploring quality measures with limited resources.

Despite the need to explore these large datasets gathered or produced dynamically, exploratory systems are still lagging behind. Progressive systems should be able to catch up with the scalability requirements for handling large data, and it is the goal of this book to speed up their development, evolution, and eventual adoption.

In data analytics, two fundamental concepts are in tension. One is related to the fact that data exploration is best done with an iterative loop, in which analysts interact with data through computations that return and visualize results. In response, analysts issue new commands, triggering new computations until they obtain an answer to their questions. The faster these loops go, the more effective analysts are: users behave qualitatively differently with rapid response times, allowing them to explore hypotheses more effectively [179]. The other, of course, is the fact that we have gotten better at collecting data and making it available for querying. Increasingly, we want to make data-driven decisions, and to do so flexibly without the need to pre-aggregate and pre-compute results. This “cold start” problem can make for slow computation.

A number of obstacles can stand in the way of interactivity. Some data sits on the other side of a network and can take a long time to move, process, and visualize. At other times, data volumes are so large that visualization can become slow. Finally, some computation algorithms take a long time to run and get longer as they become more sophisticated. The purpose of PDA is to mitigate these obstacles.

1.3 Illustrative Scenario

How would a data analyst perform Progressive Data Analysis assuming the existence of a full PDA system? We could build on multiple experimental systems, merging their best features to imagine a mature product. There are currently two very different approaches to exploratory systems. Data scientists tend to use computational notebooks such as Jupyter for Python and RStudio for R, while domain practitioners tend to use graphical applications such as Tableau or Power BI. PDA can be used in either of these approaches with proper modifications to the user interface as outlined in Chapter 5 Visualization.

As an example scenario, consider the role of data exploration in public health. This scenario is adapted from a collaboration with French Social Security for the analysis of patient treatments in France. The goal is to update national recommendations for specific diseases using evidence from massive amounts of medical treatment data.

The Ministry of Health has a strong interest in understanding the effects of public health measures, including by searching for both cost and utility efficiencies. To that end, it asked Social Security to start analyzing data initially collected for reimbursement purposes for all persons covered by French Social Security—about 70 million at any given time—with a history of 25 years.

Unfortunately, the Social Security researchers are extremely limited in how they can explore and interact with the data. The data presents challenges both in terms of scale and form. There might be hundreds of millions of patients visiting doctors and hospitals in any given year. An individual patient might be associated with thousands of records and each record might contain hundreds of different fields. The data is heterogeneous: an X-ray might be associated with a radiologist's analysis, imaging, and diagnostic code; a prescription might be associated with the dosage, frequency, and so on. In addition to structured data, an analyst may need to examine unstructured text, images, and even audio or video data.

The process of working with the data is arduous. When a medical data analyst wants to explore this data, they will often write a series of data cleaning and transformation scripts that move a subset of the data into a smaller database, which is then pre-indexed. The disadvantages are clear: the analyst is limited to making decisions about the data before cleaning and transforming, and the costs of this work make it impossible to explore new questions quickly.

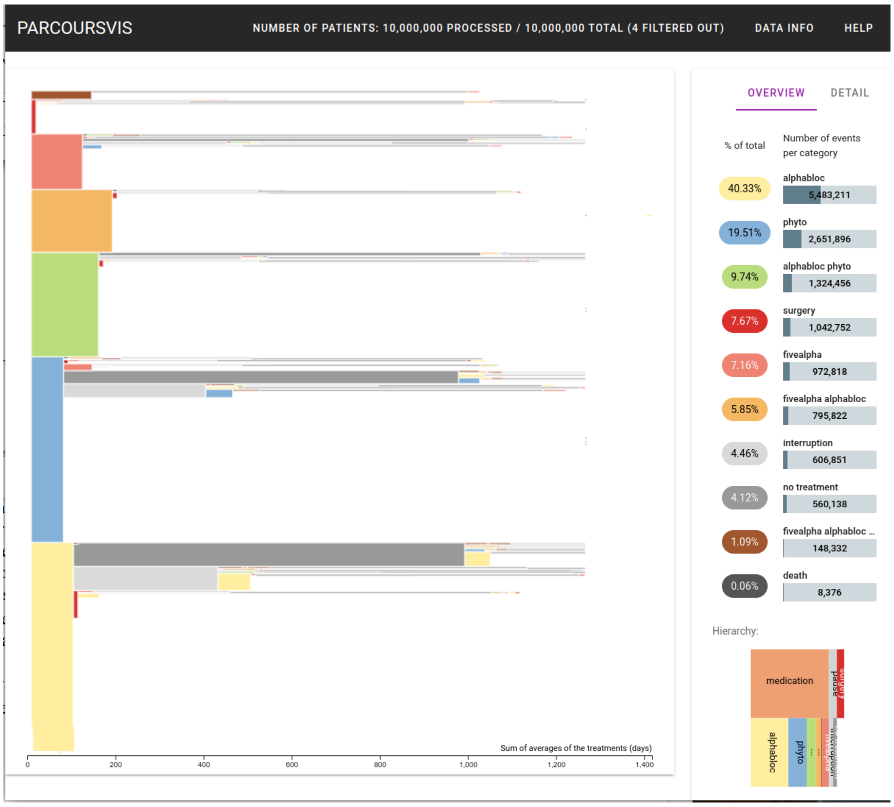
The promise of PVA is that it can allow complex data to be explored at scale by the medical doctors with a PVA Health Explorer. A group of specialists in charge of national recommendations for a disease or condition can visualize the aggregated treatment history of all the French patients. Many chronic illnesses involve millions of patients over decades—billions of medical events.

Currently, systems for exploring these aggregated patient pathways are in the experimental stage [201], but they are limited to 50–100k patients with hundreds of events. A progressive system can easily handle hundreds of millions of patients with thousands of events, allowing doctors to look at the situation at the country level. The first thing the doctors will want to understand is how the drugs are being used. They will do this by looking at an overview of the treatment pathways (see Figure 1.1).

The overview is computed progressively, meaning that the beginning of the aggregated history stabilizes quickly, while the deeper leaf nodes may take a few seconds to stabilize. Fortunately, the doctors are initially interested in the main trends that are near the beginning of the treatments. They notice that in about half the cases the treatments follow their recommendations and are consistent with their expectations. But in the other half, there are unexpected situations. For example, they believed that after surgery all the patients would be cured. But in reality, there will always be a certain percentage of patients—about 5%—who need continued treatment, sometimes including a second surgery. Why does that happen? All of these unexpected situations call for further exploration and steer questions. Is it an issue of old recommendations vs. new, older patients vs. younger? Does it have to do with patients with a particular set of health conditions (diabetes, hypertension, overweight, etc)? By filtering for the large number of attributes associated with the patients, the doctors can try to discover why some patients are not following recommendations. Selecting a rectangle shows the associated attributes to the right. The system allows filtering using any number of attributes or history patterns to limit the visualization to specific contexts and conditions. A change in the filter will update the visualization progressively, providing an approximate result immediately, with details becoming more stable in a few seconds at most.

The doctors are able to use these filtered results to collect a set of unexplained cases. They can build a cohort of patients for further analysis. Again, even while the analysis continues, they have a first few patients that they can check manually. For example, these early samples might show that there are a lot of patients with this diagnosis being treated in a very different way because they suffer from high blood pressure. Perhaps there are interactions between the drugs being taken for the two conditions, leading the doctors to modify their recommendations on that basis.

At each step of the exploration, the ability to view a fraction of data allows doctors to decide whether to use the current result or to continue exploring. In all cases, progressiveness provides them with partial results in just seconds, regardless of



% of total	Number of events per category	
40.33%	alphabloc	5,483,211
19.51%	phyto	2,651,896
9.74%	alphabloc phyto	1,324,456
7.67%	surgery	1,042,752
7.16%	fivealpha	972,818
5.85%	fivealpha alphabloc	795,822
4.46%	interruption	606,851
4.12%	no treatment	560,138
1.09%	fivealpha alphabloc ...	148,332
0.06%	death	8,376

Figure 1.1: Icicle tree visualization of the aggregated events of 10 million (synthetic) patients treated for non-cancerous prostate adenoma. Each colored rectangle represents a treatment; its height is proportional to its frequency (e.g., the large yellow lower left rectangle's height represents 40%, meaning that 40% of the patients started with the drug "alphabloc"), and its width represents the average duration of the treatment, except "surgery" and "death" that have no duration. On the right of the yellow rectangle is the next stage of the treatment. For the most part, people who started with it are staying with it until the end (the large bottom right section after the rectangle is empty). About 5% had a surgery (red line) or stopped their treatment for a certain time before continuing it or using another molecule.

data size, allowing them to stay engaged in the exploration and leading to possible enhancements to the treatment recommendations.

1.4 Resistance to Progressive Data Analysis?

PDA's detractors are concerned that PDA systems have requirements that are incompatible with most existing software libraries. Traditional libraries have APIs that are based on function calls that run once, not progressively. A traditional function call may take an unbounded amount of time to run, but returns a single result that is as accurate as possible. Calling the equivalent progressive function, as discussed in Section 2.3, would return a sequence of results, each bounded in time and improving in quality. Such a paradigm shift would be expensive to achieve, so it is worth considering whether there might be alternatives to PDA.

Scalability in (visual) exploration can be achieved without PDA when the dataset and exploration tasks are well defined, allowing resources to be invested upfront to match latency constraints. A number of systems are scalable because they perform pre-computations to serve results without performing overly expensive (and time-consuming) calculations [133]. The Nanocubes system [177] allows exploration of large spatiotemporal datasets by pre-aggregating results ahead of time. While the pre-aggregation step can take hours, the system runs in interactive time afterward. The Cartolabe system [39] allows visualization and exploration of large textual corpora by pre-processing them: downloading, analyzing, and performing expensive computations (NLP, multidimensional projection, clustering) ahead of time. The Falcon system [206] allows exploration of long time series using cross-filtering by relying on a high-performance approximate database, requiring a preparation time to ingest and index data. These scenarios do not apply when exploration is performed only once and has to be cold-started; the initial investment of collecting data, cleaning, and pre-processing it must be accounted for as bottlenecks or even roadblocks.

By contrast, a pure PDA system allows cold starting of the exploration process: there is no need to download datasets before beginning exploration, no need to pre-process data or create indexes. PDA systems reduce preparation time and enable spontaneous explorations, in comparison to those that involve significant preparation overhead. *For exploring large data with complex algorithms, open-ended tasks, and bandwidth-limited data communication channels, PDA has no clear competitor.*

1.5 A History of Progressive Data Analysis

Early solutions using progressive computations date back to the late 1990s with Hellerstein *et al.* [117] coining of the concept of "online aggregation". This term is still used in the database domain to refer specifically to the output aspect of a progressive computation, albeit without steering. Although early articles on online aggregation also mentioned visualization, the progressive approach has specialized in different directions, and has relied heavily on visualizations during the first decades of the new millennium. These have been introduced under different names, such as Incremental Visualization [89], Progressive Visualization [92], Per-Iteration Visualization [157], and Progressive Visual Analytics [11]. The idea of displaying partial results before the final result is complete was not unique to database researchers. The graph-drawing community has been using progressive layout methods since the early '90s following Eades' work on the force-based layout method [74]. Early Web browsers (Mosaic, Netscape) of the same era also featured progressive download of web pages, images, and progressive HTML layout as a way to mitigate the Internet's latency. These early progressive systems were designed for very specific tasks, whereas PDA is aimed at general data exploration.

There is as yet no unique consensus name for these approaches, and this book uses the term *Progressive Data Analysis* as a generic term, along with others such as *Progressive Visual Analytics* (PVA) to emphasize the role of visualization and interaction. Where other terms appear in the following chapters, they are relics of their various origins. One of our goals is to connect previous studies related to PDA in an effort to strengthen the field, identify the research questions raised in the past, and collaborate in order to answer them both now and in the future, crossing over traditional research domains.

In Chapter 2, we'll outline some of the philosophical differences between approaches that are somewhat similar to PDA, highlighting the main differences and addressing possible sources of misunderstanding about PDA.

1.6 Goal of the Book

The goal of this book is to tame this proliferation of paradigms and approaches related to Progressive Data Analysis.

In the past, attempts to build progressive systems have been slowed down because of the fragmentation of the research fields. Database researchers have been able to build early progressive database engines [116, 117], but their results were not usable in traditional back-ends. The visualization community also designed early progressive systems, but their data management has not proven scalable. Lying somewhere in the middle, data analysis and machine learning, despite their needs, have only been able to build prototype progressive systems [293, 334] due to the lack of adequate data management and visualization support. We strongly believe that the problem of Progressive Data Analysis should be tackled in a holistic way by all three communities working together, as is reflected by the chapter authors of this book. This will allow the field to grow and mature instead of remaining forever stuck in its infancy.

Data analysis researchers and practitioners have already begun this process of software evolution toward progressive systems. In October 2018, they gathered for a seminar in Dagstuhl to exchange ideas, experiences, and visions [82]. They are all convinced that in the coming years, Progressive Data Analysis will become a leading paradigm for data exploration systems, but that it will require major changes in the algorithms, data structures, and systems of today.

Most of the existing research in PDA has been done within one computer science community, sometimes in collaboration with a particular application domain such as medicine and biology (e.g., [128]), but rarely in collaboration with other communities or domains.

The Dagstuhl seminar discussions convinced us that these silos are largely responsible for the slow development and adoption of PDA, and the goal of this book is to help break through the barriers standing in the way of this work and to foster collaboration.

This book summarizes the main challenges that need to be addressed in designing progressive systems and lays out a scientific roadmap for attacking the problems in a constructive and consistent manner. It is, therefore, meant to serve as a first step in this process, to clarify for all of the research communities what the main issues are that need to be addressed, why they are important, and why they are sometimes difficult. It seeks to provide a useful roadmap that can be used to find solutions to the main challenges, deliver usable PDA systems to support data exploration, and make sense of current and future data.

1.7 A Research Future for PDA

PDA presents many exciting research opportunities, which are discussed in Chapter 10. They will also be mentioned as they come up in the book; research agenda entries are introduced as shown here: **Research Agenda:** (note the icon in the margin).

In our earlier example of the election scenario in Section 1.1, we saw some of the major issues that arise in progressive computation. Partial results had to be measured with wide error bars. To interpret the results, the news-watcher or analyst must understand the biases of the underlying system—different voting habits related to vote-by-mail vs. in-person ballots. There were even implications associated with different state-wide policies affecting when ballots could be counted and what data was available.

When designing a PDA system, it is vital to ensure that there are ways to help the user estimate current progress, recognize potential sources of bias, and keep them from prematurely reacting to incorrect results, and when they do make that mistake, to help them recover from it. Unfortunately, it can be hard for people to correct past errors.

We will frequently return to the question of how to manage the trade-off between stability and fresh results, ensuring that users are able to track how the results of partial computations change over time, especially in Section 5.2.1. Traditionally, visualizations have been built based on the data at hand. PDA implies that we will need to keep updating the visualization, but radical changes in scale or size can be disorienting.

For example, progressively loading and visualizing a network graph can be reasonably stable. When a node is added, a physics-based layout can often do a good job of maintaining relative positions around it. Connecting two existing nodes can change the layout quite a bit, especially if two disconnected components of the graph become connected; the graph layout then changes dramatically. Avoiding or overcoming these abrupt changes is a challenge for PVA. Other visualizations are even more sensitive: visualizing a bar chart while loading a large database progressively is sensitive to small data changes. For example, when visualizing the height of the average person in each country sorted by size, if two countries have almost the same average height, their positions may flip randomly when a new chunk is processed. PDA must find new visual or interactive strategies to avoid or mitigate these insignificant and distracting changes.

We will also return to the question of whether partial results are meaningful in Chapter 7. It can be tempting and risky for a user to quickly respond to a non-

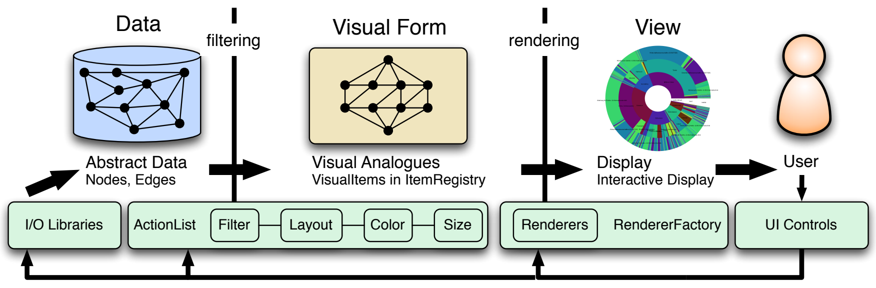


Figure 1.2: The Information Visualization Reference Model, adapted from [41]

representative sample of the source data.

There are challenges related to choosing progressive algorithms for machine learning, as we address in Chapter 8. In modeling, simulation, and forecasting, it is important to have humans monitor and steer progress. It can both ensure that the ML system obtains a meaningful result and can save power by avoiding expensive model training when the outcome can be foreseen by a skilled human.

This book calls for collaboration between the database, visualization, HCI, psychology, and machine learning communities, as well as others, such as simulation. It is intended to help bridge the gap between these communities and accelerate the development of common ground to facilitate such collaboration.

1.8 Organization of this Book

As visualization researchers, we tend to think about PDA in the context of the information visualization reference model [41] (Figure 1.2) and the Visual Analytics reference model [154] (Figure 1.3). The Visualization Reference Model shows data going from a data source through a series of transformations and into a final view. The Visual Analytics Model suggests that data goes through a visualization phase, as well as data modeling and mining. The two models remain valid for PDA, but data processing and visualization are now progressive in both models.

The outline of the book follows these two reference models (Figure 1.4): from data management to users, including perception and mental models, and from the connections between visualization and data modeling and mining. The key information that should be taken from the reference models comes by recognizing the dependencies

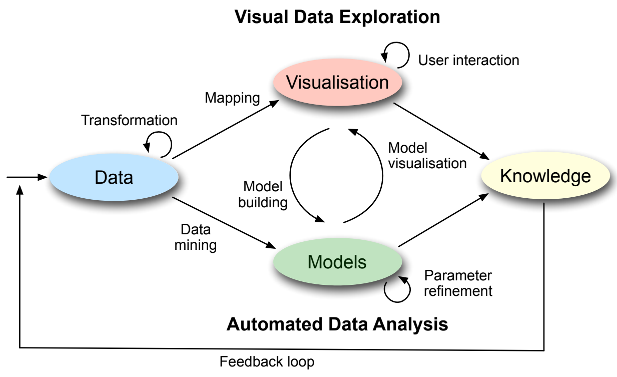


Figure 1.3: The Visual Analytics Reference Model, adapted from [154]

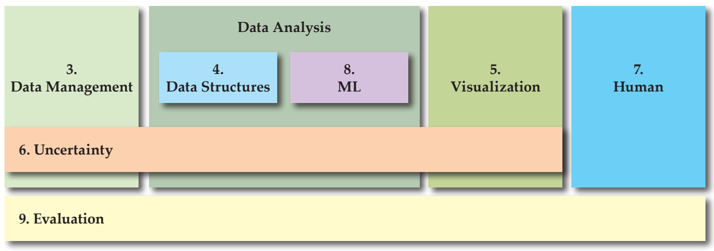


Figure 1.4: From left to right, logical organization of book chapters related to the data analysis pipeline.

between the areas. In traditional EDA, we can often abstract away different domains. That is, as long as there is a data table available, we can find a way to visualize it. The design of the data management and mapping algorithms will influence the sorts of visualizations that we can create. Requirements from user perception and cognition will influence requirements from data management on the other side of the pipeline. Therefore, the development of PDA solutions requires an awareness of the entire reference model and pipeline, overcoming the traditional separations.

We start by defining what PDA is in Chapter 2, comparing it to other well-known execution paradigms such as *streaming computation* and *anytime algorithms*. Throughout the book, we contrast progressive algorithms and visualizations with their traditional counterparts, which in this book will be designated with the word *eager*.

We then introduce the typical issues that arise with a complete PDA sequence of operations and interactions, starting with data management (Chapter 3) for loading, storing, indexing, and querying data in a progressive way.

We then discuss how general computations are applied (Chapter 4) as data is progressively loaded into memory. We discuss how classical data structures, arithmetic, linear algebra, and statistical methods can be adapted so that they are progressive.

Because Progressive Data Analysis is aimed at data exploration, visualization and the user interface play major roles. Chapter 5 addresses the main questions raised by the evolution of visualization techniques and the need to support user interfaces for progressive results in a way that is compatible with human perception and cognition. With progressive methods, visualizations should become scalable to very large amounts of data. This will also raise new scalability issues, some of which are specific to the use of progressive algorithms.

Chapter 6 addresses the problem of uncertainty with progressive computations that generate approximate solutions that improve over time. In these cases uncertainty needs to be assessed and presented to the analyst in an understandable way to assess whether a decision can be made at each point.

This issue of decision-making is one of many such topics discussed in Chapter 7 that touch on the human aspects of PVA and understanding.

Chapter 8 is dedicated to applications of progressivity to machine learning. Some methods are inherently progressive, some can be made progressive with adaptations, while others are more challenging.

Evaluating progressive systems is addressed in Chapter 9. This chapter discusses technical evaluation methods that are traditional in the database and data analysis communities, but also human-centered methods that are closer to the visualization and HCI communities. Both kinds of methods should be considered to ensure that progressive systems are effective for data exploration at scale.

The book concludes with a summary of the challenges raised by Progressive Data Analysis and offers a research agenda (Chapter 10) that should be approached in a multidisciplinary manner. Such an agenda will require several years of research and experimentation to reach a point of maturity where it offers high gain for high risk, just as all the paradigm shifts of the past have done.

2 Concepts and Definition

2.1 Motivation 16
2.2 Concepts 18
2.3 Definitions 22
2.4 *Progressive-in-chunks* and *Progressive-in-iterations* 30
2.5 Summary 31

Authors Giuseppe Santucci and Michaël Aupetit

2.1 Motivation

When exploring data, the computation of meaningful results requires that data be available in sufficient quantity and quality, and that sufficient time be allowed for the computations to finish. However, in many cases, this setting is not optimal, if impossible at all, for those in charge of the computation.

When data access or computation is too slow, we can use data sampling. But how much sampling is enough to obtain meaningful results? We can, of course, also resort to iterative or incremental computation techniques and get our results gradually or progressively instead of all at once. This is what Progressive Data Analysis aims for. PDA can therefore be described as a data analysis technique that produces partial results during execution until the final results are computed.

As noted in Section 1.5, there is not yet a unique, agreed-upon name for this technique, but in this book, we call such techniques Progressive Data Analysis and Progressive Visual Analytics. A PDA system can be described as one that:

1. produces partial results that improve with time, with controlled latency,
2. uses visualization for the presentation of results,
3. meets appropriate requirements for the progression to be meaningful.

We will define these concepts more precisely later in this chapter. We will call non-progressive systems *eager*. Eager systems compute results all at once without controlling latency.

This chapter aims to provide a common framework for defining and formalizing the ideas and techniques associated with PDA and to clarify where *progressive* methods stand in the landscape of other related methods.

The concept of working through incremental results is not new in this kind of work. Indeed, we can see such methods as part of a wider landscape of systems, each designed to solve different kinds of problems. To get a sense of some of these specializations, we can look at some related concepts in the field of database research and other fields.

In Section 1.5, we made reference to the database concept of "online aggregation" (OLA). Database researchers have also explored the concepts of "streaming computation" and "anytime algorithms". *Streaming computation* is an approach in which new data is streaming in continuously, e.g., from a remote sensor, and the associated time plays a significant role in its analysis, i.e., it has to be analyzed using a sliding time window. *Progressive* systems, by contrast, assume that the data is **static**, and thus does not change over time. The result analysis is conducted in a series of steps intended to satisfy latency constraints. *Anytime* algorithms start a computation process on the back-end. Here, the user can query the system at "any time" to request the latest computed result, which improves as more computation is performed. The user takes the latest result. This distinguishes it from online, streaming, and progressive systems, which push updates to the user. PDA generalizes these concepts, allowing interactive control steerability, and when it includes visualization, we call it Progressive Visual Analytics (PVA).

Progressive systems can either have the goal of returning a single result or returning multiple results. For the latter, we will refer to *search/gather* progressive systems, in which the ultimate result is a list of results. In those cases, the progressive system may progress by incrementally returning results.

The proliferation of related concepts is both a blessing and a curse. It is a blessing because it acknowledges the need for a computation paradigm to help humans in exploring data at scale with complex analytical methods. There are already significant bodies of research and application work covering areas such as data streaming, online computation, and iterative algorithms. It is a curse because discussions of these concepts are often confusing and lead to misunderstandings. Is "Online aggregation" identical to PDA? Why aren't streaming methods able to solve the latency problem? Our goal in this chapter, then, is to articulate the range of possibilities, to explain how these solutions are related, and to assess some of their trade-offs.

Ultimately, we see these multiple approaches as a strength. Algorithms and data processing paradigms that are developed in one tradition can sometimes be adapted to others. For example, a clustering algorithm that can work on partial data might also be developed for use in anytime computation. It can be easily used for progressive systems, and adding knobs to make it steerable can make it even more powerful.

Key to the concept of progressive computation is the aspect of human monitoring and control. This can be illustrated by contrasting it with other approaches. In algorithm animation [155], an algorithm is instrumented to visualize the multiple steps it produces when it runs on data. In that case, the algorithm drives the system, and the human monitors the algorithm with limited control—often doing nothing but starting, pausing, and stopping the animation. In-situ visualization [185] is another important topic in visualization related to simulation. In-situ systems instrument a simulation program to generate visualizations on-the-fly during simulation to allow monitoring of the simulation and, to a certain extent, controlling (e.g., the user might pause, change a few parameters, get more details about the ongoing simulation, stop/abort, etc.). In-situ systems are algorithm-driven and not human-driven, as PDA systems are, and they have different architectures than progressive systems. For these reasons, we will not discuss them further in this book.

In this chapter, we will attempt to clarify several of the concepts that are most relevant to PDA, while providing formal definitions of the main methods.

2.2 Concepts

In this subsection we will look at the basic PDA-related concepts, while also introducing the main elements of the PDA process, detailing its different facets and analyzed data. We will also examine Progressive Visual Analytics as a more comprehensive form of PDA in terms of type of data, interaction, and visualization. In Table 2.1, we list the main requirements for PVA systems, classified along the axes of feedback, control, result, and execution. We will refer to and illustrate these requirements along with the descriptions in the text.

Progressive Data Analysis

The next section attempts to formally define what we mean by Progressive Data Analysis. A PDA system takes data as input and produces a sequence of *meaningful, partial* results (design requirements R1, R6, R7) characterized by increasing *utility*. The early *real-time* results (R3, R9) are helpful for the analysis at hand (R10, R11) and each result should be better than the previous one (sometimes after an initial

	Result	Execution
Feedback	R1: "Provide increasingly *meaningful partial results* as the algorithm executes" [293] R2: Return "*structure-preserving intermediate results* [207], i.e., results that are structurally equivalent to the final results R in the sense that the same techniques for visualization and data processing can be applied to them" [207] R3: Avoid interfering with users' *cognitive workflow* on updates [293] R4: *Minimize distractions by not changing views excessively* during updates [293] R5: *Provide cues* to indicate where new results have been found by analytics [293] R6: Return *aggregated information* that "provides a certain aspect of intermediate results R without preserving the structure in full detail" [207] R7: Return *Uncertainty* "concerning the final result R as estimated based on the available intermediate information" [207] R8: Return *provenance* "information, any type of meta-information concerning simplifications made for generating R" [207]	R9: Show *aliveness* to "confirm that the computation is in progress" [207] R10: Show *absolute progress*, "information about the current execution phase of F_p which may be qualitative (e.g., 'computing distance matrix...') or quantitative (e.g., 'iteration 12' for k-means or the number of processed data items)" [207] R11: Show *relative progress*, "information about the degree of completeness of F_p which is frequently provided as a percentage or as an estimate of the remaining time" [207]
Control	R12: Provide *full interactivity* [119] R13: Support an *on-demand refresh* when analysts are ready to explore the latest results [293] R14: Allow to *steer the final result R* [207]	R15: Allow *cancellation*, "the most important type of execution control" [207] R16: Allow "*prioritization* of the remaining work" [207]

Table 2.1: Design requirements from the related work [119, 207, 293] for creating PVA systems in terms of the user feedback and control provided for the results and executed processes (adapted from [16]).

phase), i.e., the progression should converge. In the following, we provide more details on real-time and convergence concepts and the related design requirements:

- **Real-time** Producing early results is a real-time activity, meaning that the result must be produced within some time constraints, typically related to cognitive aspects [R3] (i.e., not too often, so as not to disrupt the human analysis flow, nor too infrequently, so that the user does not lose focus) and human-computer interaction [R9] (not too late, which disrupts the interaction flow) [179]. A real-time system can be classified as soft, firm, or hard [280]: it is *hard* if the failure to meet a deadline results in total system failure; it is *soft* if the usefulness of a result degrades after its deadline, thereby lessening the system's quality of service; and *firm* if a small number of missed deadlines are tolerable but may degrade the system's quality of service. A PDA is a *soft* real-time system. It also allows *early termination*, in which the sequence of results is stopped before reaching an optimal result when a specific condition is met—e.g., the sufficient target value has been reached [R10], the confidence interval for the result is below a threshold [R11], an interaction triggered new computations with modified parameters [R12], the latest results are requested for refresh [R13], steering triggered recomputation [R14], computation has been canceled [R15] or re-prioritized [R16].
- **Convergence [R1, R6, R7, R8, R9, R10, R11]** PDA results form a sequence related to their provenance [R8]. There are multiple notions of convergence depending on the computed results. One typical case is convergence to a scalar value. Formally, a series of scalar results S_i are convergent (Cauchy convergence criterion) if there exists a number r (the limit) such that, for any small positive number ε and a sufficient number of steps N, $|S_i - r| < \varepsilon$ for $i \geq N$. We do not expect a PDA to reach a final target value r in a finite number of steps, but we expect that after a finite and *small enough* number of steps, the result will approximate the target with sufficient accuracy,[1] i.e., within a *small enough* error bound (called ε above) while accounting for its uncertainty [R7] to make these partial results *meaningful enough* [R1]. Moreover, further steps would make absolute progress [R10] by further reducing the approximation error, while making sufficient relative progress [R11] with each step, leading to a *significant* enough improvement of their utility.
- **Algorithmic Stability [R9, R10, R11]** Algorithmic stability implies that convergent results should change in small and consistent ways: additional

[1]"Accuracy" is defined by ISO 5725-1 [138] as "the closeness of a measurement to the true value"

data or additional iterations should not make major changes to the results. Stability may also depend on data: some progressive solutions split data into chunks, which are processed one at a time until all the data has been consumed (we call these *progressive-in-chunks*). Chunks must share the same properties so as not to bias the computations of the initial partial results. Formally, we say that data must be *stationary*, as described in Section 2.4. Algorithm stability is associated with the concept of Visual stability, which is discussed in Section 5.2.1. Note that bias and uncertainty [R7] can sometimes be evaluated and compensated for (e.g., by removing useless values or adding missing values). A stochastic definition of convergence can be considered an alternative [28], which focuses on another type of aggregated information [R6], e.g., the characteristics of the statistical distribution of intermediate results or some measure of probability defined by them. In search/gather (item 2.1) progressive systems, a progressive computation can also search for a list of values, like Google search does for web pages matching a query. In that case, convergence has yet another meaning related to the precision and recall of the returned list relative to the query. Intuitively, at the end of the progressive computation the results should be as close as possible to the result of a perfect *eager* computation.

Research Agenda: Questions such as what aggregated information [R6] to use; which definition of convergence to use; what is small, meaningful, significant enough to obtain converging results [R1, R7, R10, R11]; and what is the right time to maintain the cognitive workflow [R3] and aliveness [R9], are all application-dependent aspects and are part of the challenge of designing PDA systems.

Progressive Visual Analytics

The term "Progressive Visual Analytics" was introduced by Stolper *et al.* [293], who acknowledged that including progressive data analysis in the Visual Analytics context can maintain the system's interactive capability, a key point in Visual Analytics methods. In a PVA system, "progressive results can be integrated into interactive visualizations that allow users to immediately explore partial results." PVA is a progressive data analysis in which the user observes the sequence of visual results and can interact with those results in order to gain early insight and steer the computation. Below are some details on the main concepts:

- **Visualization [R2, R3, R4, R5, R6, R7, R8]** Derived data can be visualized, transforming the sequence of results into a sequence of visualizations. This

extension can introduce additional latency to the process and requires both a more complex definition of usefulness and additional care in displaying the results to the user. Consistency should be maintained throughout the sequence of visualizations [R2], the cognitive workflow should be maintained [R3], and distractions during updates should be minimized [R4]. For example, if a new result is signaled but that result produces a change in the visualization that is hard to perceive, then it is considered to be of questionable usefulness, as it can cause the user to lose focus on other important tasks to an unnecessary degree, and producing that visualization is a waste of resources if the change will not be noticeable anyway (see, e.g., [9]). In addition, this opens up the possibility of visually representing changes [R5], summary results [R6], and their uncertainty [R7] and provenance [R8], e.g., visually distinguishing stable elements in a cluster so that they can be tracked during progressive clustering.
- **Computational steering [R12, R13, R14, R15, R16]**. In terms of controlling the process, the analyst should have, at a minimum, control over starting, pausing, slowing down [R13], and speeding up, as well as stopping (early cancellation [R15], early completion). This can also include branching an analysis into two concurrent processes—e.g., running the same algorithm on different data, or different algorithms on the same data, to compare or interleave their results [R14]. Finally, more involved forms of steering include narrowing and refocusing [R16] the process on certain subspaces of interest within the dataset, or to stop [R15] the processing of data subspaces that are not relevant to the current task or analytic question.

PVA is probably one of the most demanding types of PDA and a great deal of effort has gone into fully defining its characteristics and requirements (see, e.g., [11, 16, 119, 207, 293]). Table 2.1 lists the main PVA requirements broken down by feedback, control, result, and execution. A more comprehensive list of requirements is available in Angelini *et al.* [11].

2.3 Definitions

Data analysis function The first step in characterizing the various types of computing methods related to PDA is to define a data analysis function from some data space that returns a result. In mathematics, the result does not involve time or accuracy. In computer science, a process to compute the function consumes some amount of *computation time* and uses a certain *amount of memory*. It starts with a *machine state*, and progresses through several internal states until it reaches the final state. For data analysis, we differentiate between the *computation parameters* and the data. Thus, the mathematical analysis function becomes a *computation function*.

In data analysis, we can also draw a distinction between search results and other kinds of computational results in the sense that the notions of stability, convergence, and quality of a search result (e.g., quality based recall and precision) are different than those of a computational result (e.g., quality based on accuracy).

Definition 1 (Mathematical and Computer function)**.** *The gray boxes provide a formal view of the methods and their characteristics.*
In mathematics, we define a data analysis function f on n values $X = \{x_1, \ldots, x_n\}$ yielding a result value r:

$$f(X) \mapsto r$$

In computer science, a function to compute f takes some amount of computation time *t, uses an* amount of memory *M, starts with a* machine state *S_1, and ends with the final state S_z. Moreover, for data analysis, we distinguish the* computation parameters *$P = \{p_1, \ldots, p_m\}$ from the* data *D it uses as input. Without loss of generality, we represent the data D with a set of* data tables *$D = \{d_1, \ldots, d_m\}$ that the function takes as input, and the outputs it returns as* results *$R = \{r_1, \ldots, r_o\}$. Since R can be approximate, some functions can return an assessment of their* uncertainty/quality*, which we denote by U. Thus our mathematical function f becomes a* computation *function F:*

$$F(S_1, P, D) \mapsto (S_z, R, U, t, M) \tag{2.1}$$

Now we use that framework to define scenarios that share common aspects with the notion of *Progressive Data Analysis*:

- Online
- Iterative
- Incremental
- Streaming
- Real-time
- Anytime
- Progressive
- AQP (approximate query processing).

For each of these, we will characterize the kind of *data* they use, how the data is *processed*, the degree to which a user can *control* the process, and whether or not the function will *converge*.

Online methods According to Albers [6], an online algorithm A is presented with a *request sequence* $\sigma = \sigma_1, \dots \sigma_n$ that must be served in the order of occurrence with the goal of minimizing the cost associated with the whole sequence. This happens for two main reasons: additional data is not available, i.e., “in any situation where decisions must be made and resources allocated without knowledge of the future” [153] or, closer to the book’s objectives, to provide the users with a result in a shorter time, see, e.g., [117]. The latter approach dates back to the 90s as an alternative to long computing batch queries, and was mainly used to obtain aggregate values on large datasets [116]. It typically provides an estimation of the confidence about the result, e.g., the confidence interval of a mean, and reuses the previous result to compute the new one efficiently. Online algorithms are popular in machine learning (see Section 8.5) and analytics in general, in particular for handling large or dynamic data.

Essential characteristics of *Online* methods:

- *Data*: bounded or unbounded
- *Process*: backward-looking to the analyzed data (e.g., reuse previous computed result for computing the new one)
- *Control*: little or no control over the input
- *Convergence*: possible in the case of bounded data.

Definition 2 (Online function). *An* online *function F_o is related to the way the data tables D are made available to the function: F_o is called iteratively with subsets D_k of the input. Formally, the growth of the data tables can be modeled with a partition $\mathcal{P}(D)$ of D into z non-intersecting subsets, also called chunks C_i: $\mathcal{P}(D) = [C_1, \dots, C_z], \bigcup_i C_i = D, C_i \neq \varnothing$ and $\forall i \neq j, C_i \cap C_j = \varnothing$. At step k, F_o is called with data D_k formed with all first k chunks, and the last result R_{k-1} that can be reused:*

$$\begin{array}{rcl} F_o(S_1, P, C_1, R_0 = \varnothing) & \mapsto & (S_2, R_1 \simeq R, t_1, M_1) \\ & \dots & \\ F_o(S_z, P, \bigcup_{j=1,z} C_j, R_{z-1}) & \mapsto & (S_{z+1}, R_z \simeq R, t_z \ll t_{1\dots z}, M_z) \end{array}$$

Online functions guarantee that each result R_i is accurate (although sometimes less than eager functions, $R_i \simeq R$). The total time t_z to compute R_z after R_{z-1} is shorter than the total time to compute $F_o(S_1, P, \bigcup_{j=1,z} C_j)$ which would be $t_{1\dots z} = \sum_{j=1,\dots,z} t_j$ ($t_z \ll t_{1\dots z}$), however, there is no guarantee that t_i is bounded.

Iterative methods An iterative function performs its computation through a number of internal iterations and provides a sequence of *improving results* at each iteration, although some extra work may be required to obtain the partial result in an externally usable form [262]. The total time to compute the final result is usually controlled indirectly either through a tolerance parameter or a maximum number of iterations, but none of these parameters allow analysts to predict the completion time.

Essential characteristics of *Iterative* methods:

- *Data*: bounded and digested in one shot
- *Process*: repeated passes over the same data (e.g., k-means, simulated annealing, ...)
- *Control*: tolerance on accuracy or maximum number of iterations
- *Convergence*: possible on final result.

Definition 3 (Iterative function). *An* iterative *function F_{it} always takes the whole datasets D and returns improving results R_i:*

$$\begin{array}{rcl} F_{it}(S_1, P, D, R_0 = \varnothing) & \mapsto & (S_2, R_1 \rightsquigarrow R, t_1, M_1) \\ & \cdots & \\ F_{it}(S_z, P, D, R_{z-1}) & \mapsto & (S_{z+1}, R_z \rightsquigarrow R, t_z, M_z) \end{array}$$

Iterative functions guarantee that the results R_i will converge to the final value R, $\lim_{j \mapsto \infty} R_j = R$, but the intermediary results may not improve at each step ($R_j \rightsquigarrow R$). There is no guarantee that t_i is bounded, although the tolerance and number of iteration parameters provide some control of t_i.

Incremental methods Incremental computing [247] deals with the challenge of "[computing] the new output incrementally by updating parts of the old output, rather than by recomputing the entire output from scratch."

Essential characteristics of *Incremental* methods:

- *Data*: dynamically updated
- *Process*: reuse the previous results and recompute only the ones depending on updated data
- *Control*: possible on the input
- *Convergence*: not guaranteed.

Definition 4 (Incremental Function)**.** *An* incremental *function F_{in} is such that the data D_z at iteration z is obtained by updates Δ on previous data D_{z-1} explicitly considering added, deleted, and changed elements:*

$$\begin{array}{rcl} F_{in}(S_1, P, D_1, R_0 = \varnothing) & \mapsto & (S_2, R_1, t_1, M_1) \\ & \cdots & \\ F_{in}(S_z, P, \Delta(D_{z-1}), R_{z-1}) & \mapsto & (S_{z+1}, R_z, t_z, M_z) \end{array}$$

There is no guarantee that $_i$ is bounded, although for many incremental functions, it will be.

Streaming methods Streaming methods can be considered a special case of online methods for unbounded data and mainly deal with memory usage and time constraints, as discussed in [212]: "in the data stream scenario, input arrives very rapidly, and there is limited memory to store the input. Algorithms have to work with one pass over the data (most of the time), space less than linear in the input size, or time significantly less than the input size."

Due to real-time constraints, streaming algorithms can sometimes compute rough approximations of their results, and in general, streaming algorithms are difficult to design and implement. It is worth noting that the sequence of results is the consequence of the real-time constrained sequential nature of the data stream and its limited memory, not an artifact of the process as with *iterative* or *incremental* methods. Nor is it the result of the need to anticipate a final result, as the stream may never end. In many cases, the computation performed over the course of a stream focuses on the most recent values, e.g., they operate on a *sliding time window*, not the whole data history.

Essential characteristics of *Streaming* methods:

- *Data*: stream inherent to the data, possibly unbounded, ordered by time
- *Process*: real-time, backward-looking to the analyzed data (e.g., density kernel estimation, mean, etc.); use of a sliding window, very often only a single pass over the data is possible with limited memory and computational resources
- *Control*: no control over the input
- *Convergence*: not guaranteed.

Definition 5 (Streaming function)**.** *A* streaming *function F_s only takes the chunks C_i and not the whole dataset; it is sometimes called a* delta*. The maximal size of a chunk can be limited by the algorithm.*

$$\begin{aligned} F_s(S_1, P, C_1, R_0 = \varnothing) &\mapsto (S_2, R_1, t_1 \leq T_{bnd}, M_1 \leq M_{bnd}) \\ &\cdots \\ F_s(S_z, P, C_z, R_{z-1}) &\mapsto (S_{z+1}, R_z, t_z \leq T_{bnd} \ll t_{1\ldots z}, M_z \leq M_{bnd}) \end{aligned}$$

The intermediate results R_i may be inaccurate, so $R_z \approx R$. The total time t_z to compute R_z after R_{z-1} is much shorter than the total time to compute $F_s(S_1, P, \bigcup_{j=1,\ldots,z} C_j)$, which would be $t_{1\ldots z} = \sum_{j=1,\ldots,z} t_j$ ($t_z \ll t_{1\ldots z}$). Streaming functions are real-time*, so t_i is bounded (T_{bnd}), and the memory M_i used by F_s is also bounded (M_{bnd}).*

Real-time methods Real-time approaches deal with the constraint of producing the output within a predefined time frame (sometimes called *quantum*) and are agnostic on what method is used. However, it should adapt its behavior to optimize the result quality according to the available time. But they cannot guarantee that a useful result is returned at each intermediary step (e.g., unknown time of interruption).

Essential characteristics of *Real-time* methods:

- *Data*: bounded data
- *Process*: optimize the computation according to the available quantum
- *Control*: pull model depending only on the time constraint
- *Convergence*: must always produce a result within the quantum.

Definition 6 (Real-time function)**.** *A* real-time *function F_* produces an output within a predefined bounded time T_{bnd} independent on the method F_* itself:*

$$F_*(S_z, P, D_z) \mapsto (S_{z+1}, R_z \rightsquigarrow R, t_z \leq T_{bnd}, M_z)$$

Anytime methods According to [103], "anytime algorithms are algorithms that trade performance for time. As the amount of time is increased, the quality of the output that an anytime algorithm returns increases. An anytime algorithm is supposed to return a valid solution even if it is interrupted before it ends." *Anytime* is an observational feature and shares with *Bounded time* the characteristic of not being tied to any particular implementation [347]. Both online and iterative methods can be used to implement an Anytime algorithm. Moreover, Anytime algorithms have no concept of steps or iteration. They do not *push* their results, but rather can be *queried* at any time for the latest, most accurate result.

Essential characteristics of *Anytime* methods:

- *Data*: bounded or unbounded
- *Process*: returns a meaningful answer on demand based on what has been processed up to that point
- *Control*: pull model that can be interrupted and queried at any time
- *Convergence*: quality cannot decrease.

Definition 7 (Anytime function)**.** *An* anytime *function F_* gets* queried *at any time q for the latest, most-accurate result:*

$$F_*(S_q, P, D_q) \quad \overset{\forall q}{\mapsto} \quad (S_{q^+}, R_q, t_q, M_q)$$

Progressive methods Unlike the previous methods, progressive methods are designed to deliver results for analysts at a controlled pace and with controlled accuracy. They are *real-time* methods that deliver increasingly-accurate results over large quantities of input data. PDA methods allow steering [210], meaning that the input data and computation parameters can change without always triggering a full restart. They aim to bound computation time, ensuring that new results arrive under a *quantum*. Like *online*, *iterative*, and *anytime* methods, progressive methods should guarantee results with increasing quality over time (after some point).

In contrast to *streaming* methods, the sequence of results associated with the analysis of a progressive method is not significant, and the incomplete results are seen not as a time window but as a partial result. We call *progressive computation* the use of progressive functions to run complete programs, sometimes called *pipelines*, in data analysis and machine learning.

We refer to the *quality* of a progressive result as the degree to which any given partial result is a good approximation of the final result (see also Definition 11). In some circumstances, we can compute *quantitative, bounded uncertainty* on an approximation. In many cases, however, uncertainty cannot be computed, but users still need an assessment of how good the approximation is.

Essential characteristics of *Progressive* methods:

- *Data*: bounded data
- *Process*: similar to online and streaming approaches
- *Control*: over input (order and parts of the data), delivery time, and quality of intermediate results
- *Convergence*: sequence of results improves over time; bias and uncertainty are evaluated and can be compensated for.

Definition 8 (Progressive function)**.** *A* progressive *function F_p takes the following form:*

$$\begin{aligned} F_p(S_1,P_1,D_1,R_0=\varnothing) &\mapsto (S_2,R_1 \approx R,t_1 \leq T_{bnd},U_1,M_1) \\ &\dots \\ F_p(S_z,P_z,D_z,R_{z-1}) &\mapsto (S_{z+1},R_z \approx R,U_z,t_z \leq T_{bnd},M_z) \end{aligned}$$

It delivers its results when called with increasing input data but will also work on changing input data, so there is no constraint that $D_i \subset D_{i+1}$. It provides a bounded-time guarantee T_{bnd} at the cost of a potentially low-accuracy estimate $R_i \approx R$. Moreover, progressive functions should guarantee that the results R_i converge to the real value R ($R_i \rightsquigarrow R$). Ideally, progressive functions should return an assessment of their quality U. Additionally, input parameters P_i can be modified between each call, and each call will provide an estimate of the result R_i that can be inaccurate. Progressive functions are similar to online *functions when D_i is growing, and similar to* iterative *functions when D_i is constant. They also reuse the previous result R_{i-1} to improve it and keep it as stable as possible.*

Approximate Query Processing Approximate Query Processing (AQP) is a database technique that, given a specified maximum delay, returns approximate results to standard queries, usually in a fraction of the time needed for computing exact results. It also relies on sampling and is similar to a progressive function run for a given specified time. One of its main advantages is that it behaves like a standard function, with one call returning the results. It fits well in the existing eager function paradigm and can be directly integrated into an existing analytical pipeline. Yet, while users receive a result in a bounded time, the uncertainty may be too large to make any decisions, leaving the user with the single option of re-running the AQP query with a longer delay.

Essential characteristics of *AQP* methods:

- *Data*: bounded or unbounded
- *Process*: real-time, sampling
- *Control*: time is controlled and uncertainty bounds are returned (sometimes the other way around)
- *Convergence*: convergence is not guaranteed, but a quality is returned.

Definition 9 (Approximate Query Processing Function)**.**

$$F(S_1, P, D) \mapsto (S_z, R, U, t, M) \tag{2.2}$$

AQP functions have real-time guarantees with no guarantee of the accuracy of the result R, but they also return an uncertainty U regarding the result.

2.4 *Progressive-in-chunks* and *Progressive-in-iterations*

Progressive functions, algorithms, and programs rely heavily on sampling to achieve their goal: to provide meaningful estimates to the user at a bounded pace. They mainly use two strategies: chunking and iterations. For example, a progressive function that computes the average of a very long table of values will rely on splitting the table into chunks and updating the returned average with each chunk. The "average" is easy to compute chunk by chunk using Welford's online algorithm [327]; we call this family of algorithms *progressive-in-chunks*.

To be efficient and unbiased, the chunks should be built as lists of random samples without replacement of the dataset, i.e., **shuffled**. If the samples are not *independent and identically distributed* (iid), the convergence of most algorithms will be slower. More generally, the data should be **stationary**: its unconditional joint probability distribution should not change when accessed earlier or later in the sampling process. For example, if you are computing the average temperature in NYC based on weather data sorted chronologically starting in January (i.e., not shuffled), the initial temperature estimates will be low, but they will then rise through August and eventually become too high before averaging out by the end of the year. An important concern with progressive systems is thus to make sure that data is processed in random order (i.e., shuffled). **Research Agenda:** It can be challenging, however, to make sure that data arrives in a random order in real applications (see Section 3.5.1).

Alternatively, many algorithms are iterative, producing improved estimates of their result by iterating through all of the data until a final result is obtained. For example, the gradient descent functions used in machine learning are iterative (Newton's method, stochastic gradient descent, k-means, etc.). We call this family of algorithms *progressive-in-iterations*. *Progressive-in-chunks* algorithms can always be used whether the data arrives in chunks or is available upfront, since it can be split if necessary to match the desired progressive quantum. Unfortunately, iterative algorithms cannot always become progressive when data is large, even when it

Method	Computation function	Output
Generic	$F(S_1, P, D)$	$\mapsto (S_z, R, t, M)$
Online	$F_o(S_z, P, \bigcup_{j=1,z} C_j, R_{z-1})$	$\mapsto (S_{z+1}, R_z \simeq R, t_z \ll t_{1 \ldots z}, M_z)$
Iterative	$F_{\mathrm{it}}(S_z, P, D, R_{z-1})$	$\mapsto (S_{z+1}, R_z \rightsquigarrow R, t_z, M_z)$
Incremental	$F_{\mathrm{in}}(S_z, P, \Delta(D_{z-1}), R_{z-1})$	$\mapsto (S_{z+1}, R_z, t_z, M_z)$
Streaming	$F_s(S_z, P, C_z, R_{z-1})$	$\mapsto (S_{z+1}, R_z, t_z \leq T_{bnd} \ll t_{1 \ldots z}, M_z \leq M_{bnd})$
Real-time	$F_*(S_z, P, D_z)$	$\mapsto (S_{z+1}, R_z \rightsquigarrow R, t_z \leq T_{bnd}, M_z)$
Anytime	$F_*(S_q, P, D_q)$	$\overset{\forall q}{\mapsto} (S_{q^+}, R_q, t_q, M_q)$
Progressive	$F_p(S_z, P_z, D_z, R_{z-1})$	$\mapsto (S_{z+1}, R_z \approx R, U, t_z \leq T_{bnd}, M_z)$
AQP	$F_{\mathrm{AQP}}(S_1, P, D)$	$\mapsto (S_z, R_z \approx R, U, t \leq T_{bnd}, M)$

Table 2.2: Summary of the different methods of computation detailed in Section 2.3. S stands for internal states, P for parameters, D for data, C for data chunks, R for results, t/T for time, M for memory, and U for uncertainty/quality.

is available upfront, as one iteration can exceed the quantum. Many articles on progressive algorithms describe ways to adapt iterative algorithms to chunking [142, 233], their characteristics, and their limits. **Research Agenda:** More research is needed to develop general strategies for adapting iterative algorithms to be applicable to data arriving in chunks. Some *progressive-in-iterations* algorithms can resort to *mini-batch* sampling to scale up. Instead of iterating multiple times over the entire dataset, these algorithms iterate over small *random samples with replacement* (e.g., [242, 272] and more generally Mini-Batch stochastic gradient descent [173]).

Other strategies exist in addition to these two main ones [124, 126], but they remain rare. **Research Agenda:** More research is needed to identify and categorize sampling strategies that are usable by progressive algorithms, and assess their efficiency.

2.5 Summary

The goal of this section was to present some of the main concepts, design requirements, and methods that share similarities with Progressive Data Analysis.

In Section 2.2, we presented important concepts related to data, real-time, convergence, visualization, and steering, and the design requirements of these concepts that need to be considered when designing PDA and PVA systems, as summarized in Table 2.1. Then, in Section 2.3, we proposed more formal and descriptive characterizations for *Online*, *Iterative*, *Incremental*, *Streaming*, *Real-time*, *Anytime*, and

Method	Data	Control →[R3,R9,R12-R16]	Convergence →[R1,R10,R11]
Online	**bounded or not**	no	possible if bounded data
Iterative	**bounded**	only on start	possible on final result
Incremental	**bounded or not**	possible on input	no
Streaming	stream	no	no
Real-time	**bounded**	only at specific time	only at specific time
Anytime	**bounded or not**	**at any time**	improving with time
Progressive	**bounded**	**at any time**	**meaningful at any time**
AQP	**bounded**	only at specific time	approximate

Table 2.3: Summary of the determining characteristics of each method detailed in Section 2.3. The desirable characteristics of a progressive method are in bold font. Control requires that the method return meaningful results [R1] any time it is interrupted, converging in some way [R10, R11]. There are additional requirements regarding the timing of these interruptions [R3, R9], and all types of control requirements [R12–R16] depend on such interruptions happening. Other requirements [R2, R4–R8] are related to the nature of the results, and their delivery to the user, and do not impact this classification.

Approximate Query Processing methods, and explained how *Progressive* methods differ from them. Their *computation function* forms are summarized in Table 2.2, and the main characteristics of each method and their relation to the progressive design requirements are summarized in Table 2.3.

Progressive data analysis is a complex and multidisciplinary activity, and its success strongly depends on considering all the relevant aspects in a holistic way: algorithms, including machine learning, visualization, uncertainty and, last but not least, the analyzed data, which is the topic of the next chapter.

3 Data Management

3.1 Motivation 33
3.2 Requirements for Progressive Data Management 36
3.3 Computing Progressive Results 37
3.4 Progressive Queries 39
3.5 Progressive Data Exploration Systems 42
3.5.1 OLA-RAW 42
3.5.2 ProgressiveDB 45
3.6 Extended Services for Progressive Data Management 47
3.7 Summary 48

Authors Florin Rusu, Carsten Binnig, and Chris Weaver

3.1 Motivation

Visualization is used for skimming through data quickly, searching for patterns along various dimensions, and drilling down into details. This process entails generating sequences of visualizations and the means for the user to continuously interact within and across them. Visual delays of more than 500 ms tend to decrease both end-user activity and dataset coverage due to the reduction in the rate of user interaction [179]—which is critical for overall observation, generalization, and hypothesis generation. However, if the visualizations are progressive, higher delays are acceptable since latency does not impact the user in the same way [342]. Instead, we can try to keep the progressive quantum below that threshold and, in doing so, maintain interactivity and responsiveness.

In this chapter, we outline considerations for managing and structuring data to improve progressivity and maintain interactive response times. To illustrate how these considerations might play out, we lay out some proposed extensions to SQL that would allow it to incorporate expressions of progressivity.

Limitations of Existing Systems We begin by considering the implementation of a progressive system over a traditional relational database. Existing data management solutions for data exploration, especially those built on relational databases, have severe limitations across the entire stack in such a case.

We note two different time restrictions. Before the first query is executed, data must usually be loaded or processed in the data management system. This can be a slow process and often entails creating a series of indexes or other statistics to optimize later querying. Second, each query may run slowly on loaded data.

To reduce upfront database loading costs, *in-situ data processing systems* have been recently introduced. They are extensions of external tables supported by standard database servers. Although external tables avoid loading, the system has to access the entire data for each query, which can lead to poor performance over a sequence of diverse queries. A series of adaptations have also been developed to address these challenges. For example, NoDB [3] and SCANRAW [49] are query-driven in-situ processing systems that improve their performance progressively and converge to the database execution time by loading all of the data. These systems execute SQL queries directly over raw data while optimizing the conversion process into the format required by the query engine. This optimization eliminates loading and provides instant access to data, thus allowing exploration to begin immediately. These systems also provide a dynamic trade-off between the time to access data and the query execution time by adaptively loading a portion of the data during processing. This allows for progressively improved query execution times while reducing the amount of storage needed for replication. All of these in-situ database systems are data-agnostic; as such, they cannot be optimized to discard irrelevant data early in the exploration. As a result, valuable CPU and storage resources may be used inefficiently. **Research Agenda:** We need to develop models of relevance and strategies for optimization based on them.

Existing eager database systems are also ill-suited to the workloads that result from interaction in visual exploration tools. When a dataset becomes large, computing results, even for a single visualization, can take seconds—or even minutes—creating a significant barrier to interactivity [21]. As we note in Chapter 1, one promising approach for computing approximate results is using a family of systems that estimates such results based on a sample of the data. By using a sample much smaller than the overall dataset, these systems greatly reduce the execution time of a query. These samples can be fixed-size offline samples, as used in BlinkDB [2] or VerdictDB [223], or *online aggregation* (OLA) [117], which uses progressively larger samples extracted online during query execution. These estimation techniques define a principled method of computing an estimation and confidence bounds for several SQL aggregation operators. Existing OLA solutions have strict requirements imposed by the sampling procedure. Data shuffling or random access by row are considered the standard procedure for extracting samples of progressively increasing size from a dataset. Shuffling generates a permutation of the data as a query preprocessing step such that a runtime sequential scan results in random samples of increasing size. However, shuffling creates a secondary copy of the data and requires significant processing time, even more than loading. To date, online aggregation has not been implemented in any commercial database system. **Research Agenda:** We should find ways to incorporate progressive aspects into commercial database systems.

Another obstacle is that eager query processing prohibits progressive result generation. For example, query processing can make use of so-called blocking operators, such as certain implementations of "join" (e.g., hash join) or "group-by". Such operators require processing all of their inputs before producing any output tuple. This has led to the development of non-blocking operators, such as hybrid-hash join, which generate result tuples much earlier—and do so continuously. Their functioning principle is to operate on blocks—or chunks—of tuples that are processed by all the operators in the query execution plan at once. This is identical to performing the entire query on a subset of the data, i.e., the selected blocks or chunks. While a partial result is generated, the important question is, "What is the significance of this result?" In the case of queries that produce tuples where the output order does not matter, the partial tuples are satisfactory. For queries that compute grouping aggregates, the partial tuples have to be statistically significant in order to produce a meaningful estimate of the result. This is done by taking random samples from the base tables and performing the subquery on them. The process can be repeated multiple times, with a partial result generated each time. These results are progressive only to the extent that the estimate is refined with each additional subquery executed. This will not be the case if the subqueries are independent. The operator state must be preserved across subqueries such that each subsequent subquery can "build" on top of the previous ones. Since this is not done by any of the existing database engines, architectural changes are required. **Research Agenda:** There are opportunities to design new architectures to support progressive subqueries.

Lastly, in interactive data exploration, users often want to quickly refine a query. For example, a user may supply a succession of different ranges to a filtering predicate. In current databases, each such user interaction results in a completely new SQL query that has to be evaluated from scratch. This results in a high overhead on system performance. In the typical use of a data exploration system, we can expect many queries to be closely related: a user might adjust just one or two parameters at once. That will lead to a sequence of closely related queries. Ideally, results and intermediate states from previous queries should be used to improve the performance of similar queries. An example of a non-progressive system that uses this insight is Falcon [206], which precaches related queries so that the user can modify the data interactively.

3.2 Requirements for Progressive Data Management

We are considering a data management architecture for progressive computation that aims to support interactive data exploration through visualization. At a general level, our goal is to define the requirements for an architecture akin to a "Progressive Tableau" [297]. Visualization is the next step in the pipeline, consuming the results of the database engine. Thus, the interaction between the visual user interface and the computing engine is realized through standard SQL queries that are executed over relational data, i.e., tables.

The visualization requirements for progressivity have to be included in all of the elements of the dashboard interface. Results of long-running queries should be returned progressively for interactively "continuous" display. Partial (approximate) results make sense when they are statistically meaningful. Progressive results can be materialized in an evolving time series to provide continuity. The query itself should be allowed to evolve based on the progressive results. This evolution can take the form of minor changes to the original query or the triggering of related new queries. The refinement process generates a hierarchy of queries connected through their lineage. The end-user analyst can further interact with these queries in two ways: they can monitor the execution by inspecting the progressive results, and they can steer the query hierarchy by stopping uninteresting queries and adding derived queries.

In addition to its functional requirements, a visualization design can impose strict response time constraints that are highly dependent on particular graphical widgets and how they parameterize queries. For example, the movement of a slider may elicit "immediate" response times lower than 100 ms, while the pressing of a button may increase the acceptable feedback time to 1 second. If a custom text entry is required, feedback time may be as long as 10 seconds [281]. Such "time engineering" constraints can affect the design of progressive execution by making some graphical widgets more suitable to be included in progressive visualization and disqualifying others (Section 7.1).

Given the functional specification and performance constraints, we outline a data management architecture for progressive computation. Rather than design a novel architecture from scratch, we embed progressivity in a modern relational database accessed through SQL. There are two solutions for realizing this approach. First, a middleware that sits between the visual interface and the database engine can implement progressivity by intercepting queries and splitting them into smaller, partial subqueries. This is a completely non-intrusive solution. The second approach requires changes to the database operators and tackles the situations that cannot be handled at the middleware layer. Independent of the approach, the progressive architecture has to provide support for progressive results and progressive queries. These support aspects, together with specific systems, are discussed in the following sections.

3.3 Computing Progressive Results

Based on these functional constraints, we can now look at technical approaches that address each of these issues.

Index-based sampling The XDB system [172] offers a possible approach in how it implements the WanderJoin algorithm [242] in the PostgreSQL database. XDB provides online aggregation-style estimates which improve progressively. The progressivity integration exists at all levels of query processing, including optimization and execution. A progressive query is specified, for example, as follows:

```
SELECT ONLINE SUM (l_extendedprice*(1-l_discount)), COUNT(*)
FROM customer, orders, lineitem
WHERE c_mktsegment='BUILDING' AND c_custkey=o_custkey AND
      l_orderkey=o_orderkey
WITHINTIME 20000 CONFIDENCE 95 REPORTINTERVAL 1000
```

Listing 3.1: Example: SQL Progressive Query

This online aggregation query tells the database engine to report estimations and associated confidence intervals, calculated with a 95% confidence level, with a progressive quantum of 1,000 milliseconds, for up to 20,000 milliseconds. In this case, the partial results—consisting of an estimate and its confidence intervals—are managed by the database engine and pushed progressively to the visualization client.

Another alternative is to allow the client to pull new results by itself. Rather than always transferring the complete result, it is also possible to send only the difference, i.e., *delta*, from the previous result.

To execute the query efficiently, B+-tree indexes can be built on all of the attributes having selection predicates and on all of the join attributes. This allows for all of the blocking operators to be transformed into non-blocking operators; thus, `GetNext()` from the query execution tree root can be pushed to a single table. Because there is an index, `GetNext()` retrieves samples from this table that satisfy the selection predicate efficiently. Joins become index lookups using the indexes built for each join attribute. As more samples are extracted from the source table, the accuracy of the progressive results increases. This requires maintaining the state of the estimate as well as the state of the aggregate.

Independent sampling Another strategy for progressive result generation is to extract samples from each of the source tables and execute the complete query over the samples. A middleware architecture is likely more appropriate in this case, with the database providing minor support. State has to be maintained by the middleware between queries for improved results. This requires efficient sampling from the source tables. In PostgreSQL, this can be done by selecting all tuples stored in the same block—information which is available in the catalog. For sampling *without* replacement, a block has to be accessed only once and this can be controlled by the middleware. To provide accuracy guarantees, the middleware can implement statistical techniques such as CLT-bounds or bootstrap.

Deterministic approximations Sampling poses drawbacks for aggregates such as maximum and minimum because no estimates and bounds can be derived from a sample; there is no tangible relationship between the maximum/minimum in the sample and the global maximum/minimum. The term "deterministic" indicates that the bounds are always guaranteed to contain the true value, unlike confidence bounds. Existing work on deterministic approximations [144, 239] demonstrates that this can be achieved by processing only a data subset—or data permutation—as long as the data is sorted in a certain order. For example, the maximum can be determined easily by accessing a single data item if the data is sorted in descending order. In the case of additional predicates and joins, more items may need to be accessed. Creating an "interesting" order is akin to extracting a sample and can be performed by creating a complete copy of the data or an index on it. The deterministic bounds can then be derived by executing a binary search operation on the "interesting" order.

3.4 Progressive Queries

Producing and visualizing the results of a single query in a progressive manner is by no means the only role of progressivity in visual data analysis. Because the results of a long-running query are produced and presented in a progressive fashion over the entire course of a query's execution, users can monitor the extent to which the evolving results are yielding analytical insights, and intervene to steer their analyses at any point. Users perform their steering actions via the respective interaction mechanisms in the visualization interface, i.e., panning and zooming, selecting regions, adjusting sliders, adding/removing attributes, etc. These actions result in modifications of the currently running query to produce the required data. Modifications to a query can include changing predicate boundaries, adding or removing filters, adding or removing result attributes, adding or removing group-by clauses, i.e., drill-down and roll-up, and adding joins for pivoting. Note that progressive joins are particularly challenging to implement correctly, as explained in several articles [106, 241, 242].

Other additions to the SQL language could expose query changes to the database engine. For instance, the language could allow queries to refer to past queries so that they can be interpreted as iterations of each other. One possible solution would be to add the ability to name queries, along with the addition of an `ALTER QUERY` statement to SQL. For example, suppose we wanted to remove the predicate on the customer table in the query introduced in the previous section—named Q1—we might write the following statement:

```sql
ALTER QUERY Q1 WHERE: DELETE c_mktsegment='BUILDING'
```

Listing 3.2: SQL ALTER QUERY

ALTER QUERY One naive approach to implementing progressive queries is to stop the running query—preferably whenever a result chunk has been produced—and start the execution of the revised query. While semantically correct, this approach does not explicitly convey the information to the database engine that the original query and its modified version are related–let alone how. Consequently, the queries are treated independently, even though they share common results. A more effective approach is to refine the previous results and devise a method to combine the new results with the old ones. Moreover, the execution of the derived query should also be optimized based on the previous queries. For instance, the modification to query Q1 given in the example `ALTER QUERY` statement could be handled by performing Q1 with the predicate `c_mktsegment<>'BUILDING'` and then summing-up this estimate with the progressive result of Q1 computed up to the query change. This choice may be optimal if the number of 'BUILDING' tuples is large, thus allowing them to be pruned away before the costly join. Another option is to perform both Q1 and its modification—essentially, a group-by on `c_mktsegment` with two groups—and then summing the results. Which of the three alternatives—the third being the execution of the modified query from the start—is more efficient will depend on many factors, such as the time when the transition occurs, the quality of the Q1 estimate, the data distribution, etc. A progressive architecture should at least be aware of all these options and consider them in a meaningful way when a query is modified.

FORK QUERY The progressive architecture we envision allows users to not only execute a single query in a progressive manner, but also to expand their data exploration by issuing additional related queries, which can also be executed in a progressive manner. In this case, the system is making sure that all the progressive queries are collectively optimized and executed. We illustrate this functionality with the following example. Consider a user who explores a census dataset and starts by issuing a progressive SQL query to get overall statistics on the gender distribution. Initial results show that the distribution is heavily skewed towards females, so the user decides to issue a second progressive SQL query to find out the distribution of females over different geographic areas. This distribution, being uniform, prompts the user to issue a third progressive query that computes the distribution of females for different age groups. At this point the system is running three progressive queries in parallel, the last two of which are dependent on the first one. Notice, though, that they do not replace the original query—they run alongside the original query. This difference is captured in SQL with the `FORK QUERY` statement, which permits more extensive changes compared to `ALTER QUERY`. Essentially, `FORK QUERY` creates a graph of query dependency relationships that must all be satisfied.

The system is responsible for creating and running a query execution plan that includes all the queries in the graph, thus optimizing overall system resource usage. This is a complicated problem because several decisions have to be made. First, the queries have to be prioritized. A simple approach is to give preference to the newer queries over the older ones. While this makes sense if we assume that the attention span of the user is skewed towards the immediate past, we may miss or delay significant trends for the older queries. The second decision is to extract the queries that can be processed concurrently and assign them the required resources. The end goal is for the user to make quick progress on their data exploration and analysis tasks, which is possible since they are able to quickly formulate and check different hypotheses without having to wait for each query to finish execution.

Fork queries enable the *parallel creation and processing of related queries*, a good strategy to conduct progressive data analysis when latency is guaranteed but the wait time to make a decision can be long.

Speculative queries The approach of progressive multi-queries can be advanced one step further by speculating on what the results of the next query might look like. For example, consider a progressive histogram over an attribute of a table in the database. An initial progressive histogram result may be coarse-grained, with a fixed number—say 2 or 10—of bins. Progressive results may consist of finer-grained bins, such as increasing from 2 to 4 or from 10 to 100 bins. The Falcon system [206] computes its bins progressively this way. We can speculate that such a finer-grained histogram will be requested and begin processing through a speculative forked query of the initial query. Since the overlap between the two queries is significant, most of the computation can be reused across them. If the database had known that the client was going to change the query in this way or a similar way, the database could have served more accurate results faster by scheduling and beginning the execution of the forked query before the client even launched the query. In the course of executing progressive queries, there is a clear expectation that subsequent queries will be made to the database. Moreover, these queries are likely to be similar to the queries before and after. This similarity may come in many forms, for example, a changed `WHERE` clause parameter, group-by attribute, or addition or removal of a `WHERE` clause. As such, the progressive architecture can "learn" the query sequence from past user sessions and predict the next queries in the sequence with high probability. **Research Agenda:** The next step is to incorporate such predicted queries into the query graph and schedule them for processing. This can be easily achieved within the progressive multi-query architecture by setting the priorities accordingly.

Rusu *et al.* [260] present a solution for generating speculative queries based on a template for model calibration using the gradient descent (GD) algorithm. While all these queries have the same structure, they are parameterized with different values for the hyperparameters of the GD algorithm. The group of queries are executed concurrently in parallel and progressively probed. The hyperparameters that are identified as suboptimal are progressively pruned and replaced with more promising values.

3.5 Progressive Data Exploration Systems

In this section, we present two progressive prototypes for interactive data exploration—OLA-RAW [50] and ProgressiveDB [23]—that implement the concepts discussed in the previous sections. While OLA-RAW provides specialized database operators inside the query execution engine, ProgressiveDB is a middleware that is implemented on top of an existing database.

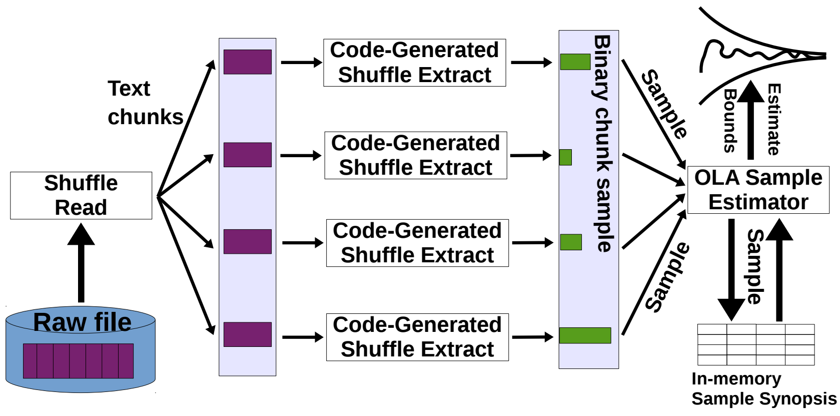

Text chunks
Shuffle Read
Raw file
Code-Generated Shuffle Extract
Code-Generated Shuffle Extract
Code-Generated Shuffle Extract
Code-Generated Shuffle Extract
Binary chunk sample
Sample
Sample
OLA Sample Estimator
Bounds
Estimate
Sample
In-memory Sample Synopsis

Figure 3.1: OLA-RAW architecture

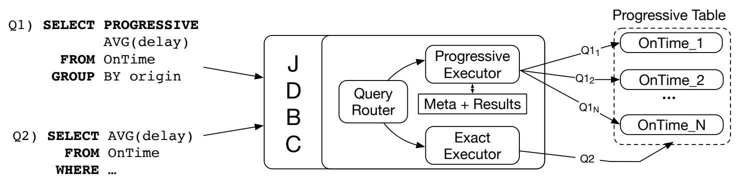
3.5.1 OLA-RAW

Figure 3.1 depicts the super-scalar pipeline architecture of OLA-RAW. Chunks of adjacent tuples are read from the raw file and passed for extraction to a pool of worker threads that are dynamically assigned work. An extraction thread converts the assigned chunk from the file format into the internal representation of the execution engine. The amount of work executed by an extraction thread varies across file formats. In order to compute an estimate and corresponding bounds, the OLA sample estimator has to be provided with a provably correct sample from the original file. Due to the asynchronous execution of the extraction threads, this is not trivial. Moreover, as more chunks and samples are extracted, the bounds of the estimate have to improve progressively. OLA-RAW carefully orchestrates the execution of each component of the system by making it sample-aware.

Bi-level sampling OLA-RAW performs estimation based on bi-level samples [107] which are a generalization of block-level sampling. Chunks are read in a predetermined random order by the Shuffle Read module shown in Figure 3.1. However, instead of aggregating all the tuples in the chunk into a single value that becomes a surrogate for the chunk, a secondary sampling process is performed over the tuples in the chunk. This is realized by randomly shuffling the order in which tuples are extracted. Independent orders are used across chunks by initializing a pair of random seeds for each Code-Generated Shuffle Extract. To cope with data structure and query variety and make sampling efficient, the code in Shuffle Extract is dynamically generated from a template corresponding to a file format, e.g., CSV, JSON, or FITS. The template takes as parameters the schema, the columns accessed by the query, and the random seeds. At each step during processing—in OLA Sample Estimator and Shuffle Extract—the set of sampled tuples correspond to a bi-level sample. This allows for estimates to be computed at any point—not only at chunk boundaries. Moreover, the sample size can be increased either by including more chunks—requiring more I/O—or more tuples inside a chunk—requiring more CPU. This provides more control over the resource utilization in raw data processing, which is known to vary significantly across file formats.

OLA Sample Estimator The role of OLA Sample Estimator is to coordinate and monitor the execution of the Shuffle Extract instances. First, it has to gather the samples from all the instances such that they form a sound bi-level sample used for estimation. Given the asynchronous execution, this is not trivial because the inspection paradox can invalidate the sampling procedure. OLA Sample Estimator addresses this issue by implementing a timing-based procedure. Each Extract thread is configured with a timing parameter that specifies when samples from the chunk have to be produced. This is achieved by forcing the Extract instances to push their samples to OLA Sample Estimator at each timing interval. Since chunks are scheduled for extraction sequentially, this guarantees that samples are extracted in order. The number of tuples included in the sample, however, can vary based on the properties of the chunk.

Second, OLA Sample Estimator has to determine the minimum number of tuples to extract from each chunk in order to optimally achieve convergence based on the available resources. This problem cannot be solved if one does not have complete knowledge of the data distribution, which is the case in raw data processing. As such, we design a resource-aware sampling strategy that dynamically and adaptively determines how much to sample from a chunk by continuously monitoring the



```
Q1) SELECT PROGRESSIVE
           AVG(delay)
      FROM OnTime
     GROUP BY origin

Q2) SELECT AVG(delay)
      FROM OnTime
     WHERE …
```

Figure 3.2: ProgressiveDB architecture

resource utilization—I/O bandwidth and CPU threads—of the system. We monitor the number of text chunks produced by Shuffle Read and the pool of threads available to Shuffle Extract each time an estimate is generated. This is the same timing parameter required to avoid the inspection paradox. As long as the number of threads is larger than the number of text chunks, processing is I/O-bound. Otherwise, it is CPU-bound. For I/O-bound processing, our goal is to end a chunk as soon as another chunk becomes available and there are no threads in the pool. In the case of CPU-bound tasks, the goal is to finalize a chunk as soon as accuracy is satisfied. We achieve these goals by carefully controlling the timing parameter at which samples are produced.

In-memory Sample Synopsis We build the in-memory bi-level sample synopsis following a process similar to reservoir sampling [58,317]. The chunks are considered for insertion in the random order in which they are extracted for estimation. As long as the memory budget is not exhausted, all the tuples extracted from a chunk are added to the synopsis, organized based on chunks. Complications arise when the memory budget is exhausted. In this case, we propose a variance-driven insertion strategy. The memory budget is divided across the chunks in the synopsis proportionally to their local variance for the current query. The larger the internal variance, the more synopsis space a chunk gets. This requires dropping some of the sampled tuples inside the chunk while preserving a sample of smaller size. By following the variance-driven insertion strategy, the synopsis is guaranteed to contain a bi-level sample at any time instant. This is important because processing can finish at any moment. In the extreme case where all the chunks are included, the synopsis degenerates into a stratified sample. For this type of sampling, we can determine the optimal order in which to extract chunks in a subsequent query—the decreasing order of the chunk variance.

3.5.2 ProgressiveDB

The goal of ProgressiveDB [23] is to enable progressive query processing and steering of queries without changing the underlying DBMS. Therefore, ProgressiveDB utilizes a middleware-based approach that provides progressive query capabilities to applications via a standard interface such as JDBC. This middleware handles all query routing, intermediate result processing, and manages the internal state of the progressive query execution as depicted in Figure 3.2. To support progressive query execution, a table has to be specifically prepared and made available to ProgressiveDB as a chunked *Progressive Table* (depicted on the right side of Figure 3.2). What is important is that chunking can be typically implemented by simply using the partition functions of an underlying database.

As a query interface, ProgressiveDB supports both standard SQL queries like $Q2$, which are directly executed on the underlying DBMS on the whole table, and progressive queries as shown for $Q1$ formulated in our Progressive-SQL language (which is a SQL extension). Both are handled by the *Progressive Executor* of the middleware. For executing a query $Q1$ formulated in our Progressive-SQL language, the Progressive Executor translates the incoming progressive queries into a series of sub-queries $Q1_1 - \cdots - Q1_N$ on the prepared Progressive Table. Each sub-query is executed on the underlying DBMS on a per chunk basis—typically a single partition of the table. The Progressive Executor progressively combines the partial results over individual chunks with the current state of the query execution. Each time a new sub-query has finished, the approximate overall result is updated and sent to the application. Consequently, the more of the table that is processed, i.e., the more sub-queries that have finished, the lower the relative error.

The middleware is also responsible for providing efficient execution strategies for query steering. For example, part of the SQL extensions are so-called *progressive views* that allow an application to refine an already running query with additional filtering predicates or to drill-down into its grouping categories. In such cases, the ProgressiveDB middleware manages all the necessary intermediate states needed for the refinement without having to re-run any already completed (sub-)query.

Database Preparation In order to be used by Progressive-SQL, a table that resides in an existing DBMS has to be prepared for progressive queries. ProgressiveDB partitions tables into chunks holding data such that each partition is small enough to be queried by an analytical query within a given query latency. An important aspect of this preparation step is that tuples are assigned randomly to each chunk so that the individual sub-queries sent by ProgressiveDB access a random sample of the table.

Note that chunking a table can be implemented without the need to physically split and copy the data of a table in the underlying database system. Instead, if a database system natively supports partitioning (as most commercial DBMS and PostgreSQL do), ProgressiveDB makes use of this capability. The table is partitioned into n randomized chunks such that the average response time of a simple aggregation query is below a configured threshold latency. Note that, for most systems, partitioning is natively supported. For systems without partitioning support, a chunking column and an index on that column are used by ProgressiveDB.

Progressive Queries With progressive query execution, clients continuously receive approximate query results, which get progressively more accurate in approaching the final result. Clients initiate a progressive query by adding the keyword `PROGRESSIVE` to an aggregation query:

```
SELECT PROGRESSIVE <aggregation_list>
FROM <table>
WHERE <filter>
GROUP BY <group-by-atts>
```

Listing 3.3: SQL PROGRESSIVE Query

ProgressiveDB translates such a progressive aggregation query Q internally into a series of sub-queries Q_i such that each sub-query accesses only a single chunk (partition) of the prepared progressive table. The results of each sub-query Q_i are then combined in the middleware with the current state of the progressive query execution to produce an approximate overall aggregation result at stage i.

ProgressiveDB computes the aggregation approximation based on the combined result of all previous sub-queries Q_j ($j < i$) and the additional partial result of sub-query Q_i. This is possible for all decomposable aggregation functions, such as `SUM`, `COUNT`, and `AVG`, where the result can be computed by aggregating over sub-aggregates or auxiliary aggregates for subsets of the data.

Query Steering During interactive data exploration, users often want to quickly refine a query, for example to further filter a result or drill-down into a result. While the progressive queries, as introduced above, provide fast (low latency) and progressive feedback to users, they do not allow the user to steer the query at runtime. With our SQL syntax thus far, refining a query would require a new query for each step, which would have to be processed by the underlying database from scratch, resulting in both execution overhead and a delay in achieving a result accuracy comparable to the already running query.

To facilitate the steering of progressive queries, Progressive-SQL introduces the concept of a `PROGRESSIVE VIEW` for which some `FUTURE` grouping attributes and filter conditions can be specified:

```
CREATE PROGRESSIVE VIEW <view-name> AS
    SELECT <aggregation-list>
    FROM <table>
    WHERE <filter-1> (AND FUTURE <filter-i>)*
    GROUP BY <group-by-atts-1> (, FUTURE <group-by-atts-i>)*
```

Listing 3.4: SQL PROGRESSIVE VIEW and FUTURE

The general semantics of a `PROGRESSIVE VIEW` is the same as with a progressive query: ProgressiveDB executes a series of sub-queries Q_i such that each sub-query accesses only a single chunk table of the prepared database schema and progressively combines these partial results with the current query execution state into new result approximations. The executed sub-queries, however, include additional grouping attributes for each `FUTURE` clause from the view definition.

3.6 Extended Services for Progressive Data Management

When all the data processing needed by an application can be done inside a progressive database, the mechanisms described above are sufficient. For more complex analytical operations or detailed visualization, however, a large amount of data will need to be transferred progressively from the database to the analytical system. For example, when computing the multidimensional projection of a data table using the *t*-SNE algorithm, all the considered tuples should be moved from the database to the analytical system. Similarly, when visualizing a table as a scatterplot, all the rows should be transferred with a selection of columns mapped to the x, y axes, and possibly other columns mapped to additional visual channels. (We will discuss more constraints on visualization in Section 5.1).

In such cases, the database can be considered to be in streaming mode, in which no aggregation or grouping is performed by the database, and all the processing is performed progressively by the analytical system. The database system can nevertheless provide some support. For visualization, knowing the ranges of data values (minima and maxima) is necessary to scale those data values to ranges of visually mapped values. Similarly, knowing the distribution of data values allows their transformation through the visual mapping to, e.g., use a log scale for the *x* or *y* axis. However, this information is not always available instantaneously. In the case of a complex query, computing the exact bounds and distribution in advance can take time, introducing latency.

Sending approximate bounds and distributions progressively is faster than having the visualization system recompute them while streaming data. However, asking a progressive database to send the bounds progressively while streaming the data should be done with care, using a fork query when one query will be I/O-bound (streaming the data) and the other CPU-bound (computing the bounds and distributions). Otherwise, the approximate bounds might not be synchronized with the data sent, and the analytical system would not visually map the data correctly for display in the visualization.

Research Agenda: More experience is needed to understand the level of support required between the progressive database and the analytical operations needed to avoid redundant work while freeing the analytical front-end of operations that are cheaper to do in the database.

Research Agenda: Many large databases are already available but are not progressive. To use PDA over the network to explore them, more progressive services are needed to provide important operations, including progressive (min, max) for a query, approximate distributions, distinct-values counts, quantiles, and possibly different levels of shuffling to trade speed for quality, to name a few.

3.7 Summary

In this chapter, we have presented progressive data management techniques for interactive data exploration and visual analytics. While such techniques have been developed for more than two decades, their adoption in practice is rather limited. One possible reason for this is the lack of an adequate interface to access and query data progressively, particularly for use in visualization applications. With a view to addressing that lack, we have focused here on concepts that support better integration between progressive data processing and visualization. We discuss these concepts in detail and illustrate how they are implemented in two prototypes for interactive data exploration: OLA-RAW and ProgressiveDB. The capabilities of these prototypes represent just a limited exploration of what could be a much larger and richer design space. **Research Agenda:** Considerably more work lies ahead to realize the design and implementation of a “Progressive Tableau”.

4 Data Structures and Algorithms

4.1 Vision . . . 51
4.2 Motivation . . . 51
4.2.1 Streaming and Steerable Visualization . . . 52
4.2.2 Creating a Progressive Algorithm . . . 53
4.2.3 Uncertainty on Aggregated Results . . . 54
4.3 Basic Data Structures for Progressivity . . . 54
4.3.1 Vectors, Multidimensional Arrays, and Tensors . . . 55
4.3.2 Data Tables . . . 56
4.3.3 Networks . . . 56
4.4 Basic Algorithms . . . 57
4.4.1 Broadcasting Operations . . . 58
4.4.2 Linear Algebra . . . 59
4.4.3 Statistical Operations . . . 60
4.4.4 Data Sketching . . . 60
4.5 Input and Output . . . 61
4.5.1 Detailed Input . . . 61
4.6 Specific Progressive Services and Strategies . . . 64
4.6.1 Progressive Stability and Normalization . . . 64
4.6.2 Data Change Management . . . 65
4.7 Summary . . . 67

Author Jean-Daniel Fekete

In the previous chapter, we examined low-level data management operations for supporting visualization and analytics front-ends. In this chapter, we move up a level of abstraction. Many data analysts today work in higher-level scientific languages. These popular scientific languages (SLs), such as Python [313], R [244], Julia [26], and MATLAB [189], are used to implement data pipelines to carry out analyses. They offer multiple layers of libraries, with standard data types such as *vectors*, *arrays*, *tensors*, *data tables* (also called *Data Frames*), and various types of *graphs* (aka *networks*). These data types support arithmetic operations, linear algebra, statistical operations, input and output, and other more specialized operations. These libraries and data types are fundamental to any data analysis and computation, and every data analyst relies on them.

However, these standard libraries are not currently designed for progressive computation. A progressive language or platform should offer data structures and algorithms similar to SLs to achieve the same level of service and avoid the current situation where each PDA application has to re-implement all of them. This chapter explores some of the issues and challenges involved in implementing these data structures and algorithms for a progressive approach. It discusses how general algorithms and data structures that support arithmetic, linear algebra, basic statistics, and input/output methods should be adapted to become progressive. The chapters on data management, visualization, machine learning, and evaluation build on some of the mechanisms described here. Many of the issues discussed in this chapter are speculative. Our goal is to provide tips that can help the community tackle challenges now and in the future.

In the examples used here, we will follow the conventions of the Python ecosystem for Data Science, which includes NumPy [312], SciPy [147], and Pandas [193]. Other SLs provide similar mechanisms with slightly different names and options.

For our purposes here, we assume that a progressive program is made up of one or more pipelines of connected progressive functions, as shown in Section 3.5. These run sequentially or in parallel to achieve a sequence of results, which are sent to other stages of the pipeline, such as visualization or data analysis functions. Most of the stages remain alive until the entire computation is completed, contrary to eager programs where each stage runs to completion and, therefore, finishes before the next one starts. A progressive program can be represented as a directed acyclic graph of progressive functions kept alive and run multiple times until they finish processing, and possibly continuing even after that if interactive steering allows for changing data in the pipeline. A single change in an earlier part of the pipeline produced by interactive steering can trigger new computations in another part of the pipeline.

Therefore, in addition to adapting the traditional data structures and algorithms, naive progressive computation pipelines can produce a large number of recomputations, slowing down the whole process and wasting resources. Advanced progressive computations require *strategies* to avoid triggering an excessive number of recomputations.

4.1 Vision

When progressive data analysis and visualization become standard, data scientists using their laptops will be able to start an exploratory analysis based on data from anywhere on the internet, either from flat files or progressive databases. Loading the datasets will produce statistics (e.g., min, max, histogram, average, and quantiles) and simple visualizations in at most a few seconds, which will be progressively updated. The analyst can then apply higher-level operations to the data, including cleaning the data, computing derived values, normalizing columns, fitting data to models, or computing multiple models. At this stage, the analyst will want to monitor the results of these operations using visualizations of the derived data, but also of the quality measures, computed progressively to decide which of the chosen operations seem to be converging. All these operations should be supported by the fundamental algorithms, including the quality measures that they return and that are fed to visualizations to facilitate monitoring and decision-making.

4.2 Motivation

Let's start with an example: one of the simplest analytics pipelines has the following form and can be executed in Python eagerly, but also progressively:

```
1  vector = read_data('huge_file.dat')
2  avg = mean(vector)
3  plot(avg)
```

Listing 4.1: Example: Progressive Mean

In an eager context, this is a straightforward three-line program: line 1 reads in a single file, line 2 computes a value, and line 3 visualizes the result.

But this could also be interpreted as a dataflow in a progressive program. These eager imperative constructs are mainly a syntax that can be turned into a dataflow representation internally, so a progressive language can look like an imperative language and use the same syntax while being executed as a dataflow system. Line 1 is an input operation that can be performed progressively and will grow the vector as the file is loaded in chunks. Line 2 is a function that can be computed progressively using e.g., Welford's method [327]. Line 3 visualizes the results while it is being computed, likely with some indication of the uncertainty of the mean value as is typical in progressive systems.

The result of the progressive `mean` function is different from the eager one: the progressive version should return a measure of uncertainty (e.g., a confidence interval) to be visualized by the `plot` function, while the eager version returns only a number.

This simple pipeline, applied to a few vectors to compute a bar chart, was used as the initial example of *online aggregation* by Hellerstein *et al.* [117], and for early human-factors studies by [88, 224].

Not all operations are easy to transform into progressive form. Computing the mean progressively is easy, but computing the median, as in Listing 4.2, is more complex, since in its strictest form it requires keeping all the loaded data in memory (this is called a *holistic* function [330]). There are streaming approximate implementations of the median, using a "quantile" data sketch, although the error can be bounded [152]. This, in turn, requires that the visualization should also show the uncertainty in a way that is faithful to the approximation.

```
vector = read_data('huge_file.dat')
med = median(vector)
plot(med)
```

Listing 4.2: Example: Progressive Median

4.2.1 Streaming and Steerable Visualization

As we noted with the median algorithm above, some algorithms that compute on data streams [97, 339] can be adapted to support PDA.

There are two major differences between data streaming algorithms and progressive algorithms: time and steering. In streams, time is intrinsically associated with data, whereas in progressive systems, it is an artifact of the algorithms and has no inherent meaning for data analysis. Many stream-based algorithms use the time explicitly to limit their computation to a time window or to decrease the importance of data as it ages. Progressive methods should not be sensitive to the order of the arrival of the data. In our previous examples, the order of the data in the file should have no impact on the final result, and the `read_data` function may load it in any order.

In PDA, steering requires the system to allow the user to modify the query without fully recomputing the data. Unfortunately, streaming systems do not support looking back at the data. Rather, they process transient data "on the fly" and build summarized intermediate representations, called "synopses" or "data sketches" [57, 58, 97], that compute approximate results and update their summarized state as data streams by.

Steerable progressive systems can also filter or update data interactively if the analyst decides to at any time—something stream-based algorithms and data structures are not designed for, as we can see in the example below:

```
vector = read_data('huge_file.dat')
filtered_vector = range_filter(vector, filter_specification)
med = median(filtered_vector)
plot(med)
```

Listing 4.3: Example: Steerable Progressive Median with Filtering

In this pipeline, the `range_filter` function takes its filter specification from an interactive filter interface (e.g., a range slider) that can be changed at any time. Because quantile sketches cannot be changed retroactively, this computation cannot be done with streaming techniques. In the example of the mean computation, updating the mean when data is filtered out can be done iteratively by slightly changing Welford's method and just removing the filtered values. Welford's sketch can be maintained incrementally when adding or removing items. Sketches computed from sets of values can usually be combined, but with limitations. Most sketches can be merged (union of items), but very few support differences (e.g., Welford) or intersections. For the quantile sketch KLL [152], an extension for these operations has been presented recently [345].

4.2.2 Creating a Progressive Algorithm

As described by Schulz *et al.* [270] and in Section 2.4, progressive algorithms can split their input in two different ways to manage their progressivity. I/O or memory-bound operations can subdivide the data into chunks (*progressive-in-chunks*) as in the examples above, while compute-bound operations can subdivide the computations into steps or iterations (*progressive-in-iterations*).

Ideally, all progressive algorithms should be able to handle data growing by chunks, but this can come at a high cost. Therefore, it also makes sense for progressive libraries to provide multiple variations of the same algorithm that are better suited to data arriving in chunks or all at once. The same is true for steering (as in the previous example): some algorithms will happily adapt to data changing and being filtered or updated, but most fundamental algorithms are not designed to deal with data that changes dynamically.

Research Agenda: Several strategies are possible for allowing algorithms to support incremental changes progressively. In the fortunate case where it is possible, an

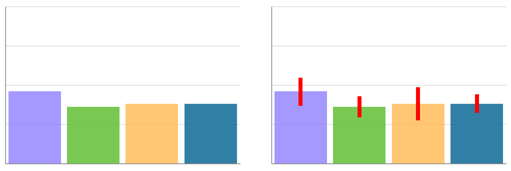


Figure 4.1: Bar charts visualizing flight delays of four companies. (Left) in a non-progressive setting. (Right) taking into account and visualizing the uncertainty of the progression [224].

algorithm implementation can be adapted to changing data. If that is not possible, higher-level strategies have to be developed to avoid complete recomputation—which is always the last resort. A third option would be the development of algorithms that are progressive in nature—they might be suboptimal in a traditional setting but better support the paradigm.

4.2.3 Uncertainty on Aggregated Results

As explained in Chapter 6, a progressive system should assess the quality of its results while computing them. Therefore, aggregated values computed progressively should also maintain a quality assessment (see Definition 11). For example, Figure 4.1 shows two designs of a bar chart, the baseline on the left and one on the right with confidence intervals to render the progressive uncertainty of the visualization.

By contrast, an eager computation of an aggregated value is exact and does not require confidence intervals related to progressivity. These confidence interval values should be computed and stored with the histogram vector. Progressive aggregated functions should, therefore, always return a value with some measure of uncertainty.

4.3 Basic Data Structures for Progressivity

All SLs support a core set of basic data structures. These commonly include vectors, multidimensional arrays, and data tables. Many also support tensors (sometimes used as a generalization of arrays and vectors) and graphs (also called networks). In this section, we discuss analogous data structures for PDA.

4.3.1 Vectors, Multidimensional Arrays, and Tensors

Progressively filling tensors SL-dense tensors are allocated as a single contiguous block of memory to support high-speed vector-based operations. Progressively populating a tensor by e.g., progressively loading a CSV file requires growing the tensor dynamically, something that is ill-fitted for one contiguous block of memory. Consider the following program:

```
array = read_csv('bigfile.csv')
derived = sqrt(array)
```

Listing 4.4: Example: Filling a Tensor

This program computes the square root of each element in the tensor. When run eagerly, only `read_csv` has to allocate an array of unknown size. The `sqrt` function knows the input array size and can allocate it up front. When this code is executed in a progressive manner, the first operation will be performed at the speed of the data transfer so that the array will grow with time, probably by chunk. The second operation will be performed on the chunks already loaded and so will require the `derived` array to also be resized as new chunks arrive. Therefore, progressive systems cannot rely directly on the existing contiguously allocated array semantics provided by traditional computing environments. Progressive systems should redesign more appropriate data representations that will grow gracefully. This is not a problem for traditional databases that handle growth; on the other hand, those databases may be less efficient at computing vector-based operations than SLs.

There are some promising approaches in existing systems that could be adapted. Several array libraries handle memory-bound computation by using tile-based allocation. Out-of-core computation on large arrays (i.e., computations on arrays that are too big for working memory) uses tile-based arrays [250, 294] and splits array operations into tiles. File storage for very large arrays such as HDF5 [56] and Apache Parquet [318] also uses partitioned tiles to support dynamic growth and updating of data. Modern linear algebra libraries such as Plasma [71] also split array operations by tiles to parallelize these operations, implementing them with multicores and GPUs. PDA could borrow insights from these tile-based systems to support progressive operations efficiently, as is done in e.g., Modin [228].

Another possible implementation of efficient growable storage is used in the MonetDB [29] database for its data columns. These columns are stored in contiguous main memory using the operating system *mmap* mechanism. This implementation allows the system to dynamically grow the column without copying the content of the array [229]. This mechanism can be used instead of tiled arrays, but it has limitations related to the resizing of multi-dimensional arrays. **Research Agenda:** More work is needed to explore different mechanisms for managing tensors in a manner that will allow for the resizing of large tensors progressively and efficiently. Existing SLs are not being implemented to manage dynamically/progressively sized tensors.

Finally, there are embeddable databases such as SQLite and DuckDB [121, 245], which can be used by progressive systems to reliably manage dynamically evolving vectors and tables. It is possible that they will also evolve to better implement the most frequent operations needed by SLs and progressive systems.

4.3.2 Data Tables

Most SLs rely on data tables internally. These data structures are well-known and well-aligned semantically with database tables (as in Chapter 3), although they can be much faster for some operations.

Like tensors, none of the existing APIs for tables are intended for use in a progressive setting. There would be challenges with dynamically growing data tables. Some operations could remain instantaneous (e.g., reading the table length), while others might need to return a stream of approximate results rather than exact ones. Other more involved operations, such as aggregation operations or joins, become similar to the online operations that are described in the database literature, with guarantees of the usability of their progressive values and quality. Some research on these operations exists, but it is still far from being readily available [117, 172, 241, 242].

Although in most SLs data tables are implemented as in-memory data structures, there is now a full spectrum of options ranging from ad-hoc in-memory to reliance on database engines. Many databases can be deeply connected to R with "connectors" like RSQLite, MonetDBLite for R, and MonetDB R Connector. More recently, database technologies have been used to re-implement Pandas DataFrames with Modin [228]. Progressive systems should similarly reimplement in-memory data tables with progressive semantics.

4.3.3 Networks

Networks are typically implemented as paired tables of vertices and edges, with integrity constraints between the tables. These networks can be implemented over progressive data tables.

Assuming the whole graph is stored in memory, simple depth-first or breadth-first algorithms to visit a graph are *progressive-in-iterations*. Many standard graph algorithms are more complex to transform into progressive form, including the standard shortest-path algorithms. To date very little research work has been done on progressive computations on networks (but see [230]).

However, there is a long tradition in network visualization and, more precisely, in *graph drawing* , of using iterative algorithms for computing graph embeddings (or layouts), starting with force-directed layouts [34, 74]. Many popular graph-drawing software can visualize graphs while the layout algorithm computes the vertex locations. To that extent, progressive visualization has been standard in the graph drawing community for many decades. In the last decade, there have been some extensions that also allow progressive computation for computationally expensive graph measures, such as Betweenness Centrality [162].

Research Agenda: Many graph algorithms are based on visitors, which are iterative but do not deliver useful partial results. For example, computing the shortest path in a graph is iterative and can be interrupted at any time, but the utility of partial results has yet to be explored. Alternatively, there are many stochastic algorithms on graphs that could certainly be made progressive, such as those based on random walks [162].

Research Agenda: There is no software package or API that provides progressive operations on networks, and the generalization of progressive data analysis on networks remains an open problem.

4.4 Basic Algorithms

Progressive libraries should be able to handle functions that are time-consuming to compute, which require a more sophisticated implementation than a traditional function. Some state should be maintained across multiple function runs, and the results should be returned along with quality and progression measures to inform the rest of the computation pipeline—and eventually the analyst. Instead of being simple functions, progressive functions become steps in a continuous computation pipeline, which can be implemented as objects with a standard interface. Transforming functions or algorithms into objects is already relatively standard in SLs. For example, the scikit-learn [226] machine learning toolkit in Python provides most of its algorithms as objects.

In scientific computing, most basic algorithms deal with tensors and data tables. We can use a popular API, such as Python Pandas DataFrames [193] as a starting list of basic algorithms. Following the Pandas documentation, for example, we can perform all of the supported operators iteratively or online: all the unary and binary operators, function application, groupBy and windowing, computation of descriptive statistics, and time series-related functions.

Some algorithms will require updating for the scale or the different semantics of progressive implementations: the issues raised in Section 3.4 will apply to implementations of reshaping, sorting, transposing, combining/comparing/joining/merging. For example, progressive sorting algorithms might return partial results with different meanings, e.g., a portion of a table is correctly sorted, the top-k elements are sorted, some selection is sorted, or the dataset is sorted within an error tolerance. Many basic algorithms can be implemented in a progressive fashion with different interpretations of an “estimate” of the result and how to assess the result quality. **Research Agenda:**A clear articulation of the semantics of the progressive versions of a function library will help build future scientific software.

Sorting and top-k algorithms are particularly interesting cases, as these algorithms can produce harmful instability in a progressive setting. Bar charts use height to visualize multiple positive quantitative values and most implementations keep the bars sorted by value to facilitate comparison. In a progressive setting, two or more bars with nearly identical values will see their estimates change slightly over time, potentially producing random changes in order over time (neighboring bars will flip). According to requirement R4, progressive systems should not change views excessively. Therefore, progressive sorting should consider the progressive uncertainty of the values and strive for stability over accuracy at each intermediary step. (See also Section 5.2.1.

4.4.1 Broadcasting Operations

Simple arithmetic operations between tensors and tables can be generalized as *broadcasting* operations [113]. An operation such as $c = 3 \times \log(a) + 2 \times b$ can be performed when a and b are tensors of the same shape, even when they are tables with an identical layout when all the operations $+, \times, \log$ are implemented on all of the columns and values. These operations are typically highly optimized in SLs, and they can be optimized just as easily in progressive computations. However, the semantics of the returned values should be clarified if the computation is not eager. If a and b are extremely large arrays, the result of $c = 3 \times \log(a) + 2 \times b$ will take time to compute and might have to be computed in multiple runs, chunk by chunk. What if an operation performed before the end of the computation tries to access a portion of c that has not yet been computed? Should it block? Or return some kind of `NULL` or `NOT_YET` value? Or should progressive systems be protected from accessing pending values? **Research Agenda:** This problem does not arise in eager computation because broadcasting operations are carried out in an atomic fashion. The semantics of progressive broadcasting operations should be well specified to deal with multidimensional tiles instead of traditional chunks.

If a progressive function is called during broadcasting, such as an aggregation function, its progressive results can be assigned, but this result will improve with time and the management of quality/uncertainty should be addressed, as explained above. The management of this quality/uncertainty is not yet well understood and will require extra storage and the computation of an aggregated quality measure to help assess the overall quality of the result (see also Chapter 6).

4.4.2 Linear Algebra

Linear algebra operations can be time-consuming to compute. **Research Agenda:** It is likely that matrix and vector products, matrix decomposition, eigenvalues computation, norms, solving equations, and inverting matrices can be done progressively, but the implementation remains to be done. The algorithms would follow a process similar to tile-based linear algebra libraries such as Dask [250], Modin [228], and Plasma [71], but they should be implemented to return meaningful information at each step. In n-dimensional data containers like tensors and tables, chunks become tiles, and these algorithms would be *progressive-in-chunks*. Current systems implementing tiled-based linear algebra ([71, 228, 250]) perform the computations in an order that minimizes data transfer and maximizes cache coherency. Ultimately, they return the full computed result and not an approximate or partial result. Progressive algorithms may choose to return only a portion of the results, e.g., for matrix multiplication, or they may perform the operation tile by tile, returning the resulting tiles in a meaningful order instead of the unpredictable order of Plasma or according to the scheduler order of Dask and Modin. Controlling that order is important in allowing additional progressive operations down the pipeline to start and progress. For example, when multiplying two matrices $o = m \times n$, the progressive results might decide to fill up the o matrix in bands in row order, column order, or some other order. This choice should probably be controlled according to the needs of the operations using the value of o. This choice may require an explicit decision from the programmer, or it might be inferred from the pipeline if a high-level PDA language is designed to do so.

Several linear decompositions and eigenvalue computation methods can be performed iteratively (*progressive-in-iterations*) or online (*progressive-in-chunks*) [43, 108, 176, 222], using exact or approximate methods. A number of stochastic linear methods are already being used in popular libraries such as scikit-learn [226] because they are faster than the exact methods. Most of these implementations hide their iterative mechanisms from the external API, meaning that they cannot be used in a progressive setting. A progressive system could copy their implementation but expose their inner state. We hope that, as progressive applications and toolkits become more popular, the standard APIs of SLs will provide progressive versions of the iterative methods that they are already implementing. Scikit-learn already offers an online implementation of PCA (Incremental PCA [255]), *progressive-in-iterations*, which has been used in some progressive data analysis applications [304].

4.4.3 Statistical Operations

Many statistical operations can be performed online and adapted to be progressive. For example, the mean, variance, and higher moments can be computed progressively using Welford's method [225, 327]. Stable samples of datasets can be used with Reservoir Sampling [317].

Clustering algorithms such as k-means perform very well in a progressive context. The mini-batch implementation [272], for example, is flexible to data being loaded progressively. It improves results progressively, which allows data to be filtered interactively, and can even be steered by changing the centroids interactively without changing its implementation. But not all statistical algorithms are as well-behaved.

For functions performing clustering and multidimensional projections, traditional SLs tend to start from pairwise distance matrices. Their computation is quadratic with the size of the input vectors, meaning that a progressive system will struggle to output meaningful results in a bounded time. Progressive variants of these algorithms could potentially rely instead on approximate k-Nearest Neighbors (kNN) [13] that are very fast to compute and use a space and time complexity almost linear with the number of items. Indeed, progressive versions of kNN are available [75, 142]. **Research Agenda:** Other solutions with different trade-offs could be explored to support the progressive implementation of important statistical operations.

4.4.4 Data Sketching

More complex statistical operations are possible with data sketching, i.e., compact data structures and algorithms that compute and maintain incrementally good and bounded approximations of values in a streaming fashion. Typical operations include quantiles, equi-depth histograms, top-*k* values, distinct value estimation, various clustering methods [97], kNN [209], wavelet computation [58], and more recently, PCA computation [94, 175], and Graph algorithms [191].

It is possible that some existing sketches can be improved to handle changes more efficiently than restarting from scratch. Jo *et al.* [142] have already shown that a data structure and algorithm to compute the *k*-Nearest Neighbors in an online fashion [209] can be adapted to become progressive. **Research Agenda:** More research will be needed to adapt existing data sketching methods to handle general progressive operations efficiently. Two adaptations will be needed: bounding their computation time (latency) and—for steerable progressivity—handling deletions or value updates in a smarter manner than complete recomputation.

4.5 Input and Output

As explained in Section 3.5.1, reading data from external sources for progressive computations offers many layers of possible optimization not provided by standard input management libraries or services. There are two different issues to address concerning input: detailed input and aggregated input. The progressive output should also deal with communication with e.g., a progressive visualization system through the web or other communication channels.

While progressive input is difficult to manage efficiently, detailed progressive output is a rather straightforward *progressive-in-chunks* process, which can use standard output streams, data formats, or databases. As for progressive aggregated output, because progressive aggregated computations need to provide an estimate of quality, writing that information to an output channel is possible, although there is currently no standard format for that. However, most NoSQL databases use the JSON format for interchange, and JSON is flexible enough to carry the extra information. Several standard file formats and protocols exist to store and transfer data, but very few can encode quality in a standard way, either with confidence intervals or other measures of quality.

4.5.1 Detailed Input

The simplest input service consists of reading data linearly by chunk, an operation offered by most standard data input systems from raw files, streams, data tables, and, more generally, from most modern SQL and NoSQL data management systems.

While streaming data progressively is possible with databases and row-oriented file formats, it is not always supported with recent file column-oriented formats, such as Arrow [12] or Parquet [318]. In that case, data will arrive column by column, limiting the options of progressive systems to manage them progressively. The Apache Arrow and Parquet formats allow chunking of data tables, but they come with no recommendation to do so. Therefore, most of these data files are currently stored in public file servers without chunking, preventing their use in progressive systems. Chunking or partitioning this storage would incur almost no overhead in writing and reading. We hope that the experience of early progressive systems and services will convince data providers to provide chunked data files.

The order of the chunks read usually cannot be controlled, and this may bias the data distribution. For example, when reading the New York Taxi dataset [214], which contains billions of taxi trips from remote CSV files, data is stored and will arrive in chronological order, exhibiting strong seasonal biases. Data will not initially be stationary, and it will require that several years of data be read to reach a stationary distribution, slowing down the convergence of any algorithm requiring stationary data.

Although some progressive algorithms have been devised to be resistant to non-stationary data arriving in chunks, they incur a non-negligible cost that slows down convergence in certain cases. For example, Responsive *t*-SNE [142] can compute a multidimensional projection progressively using an adaptation of the *t*-SNE algorithm. It detects non-stationary data and reacts to it by re-launching the “exaggeration” phase of the algorithm that conceptually reshuffles the projection. Figure 4.2 shows the results of applying the algorithm to a completely skewed dataset when the high-dimensional points are read in label order. The final projection is well recovered but requires more resources to compute and devise the algorithm adaptation to protect against non-stationarity.

There are several strategies that can be used to improve linear order reading. In the best case scenario, when the data can be processed before it is loaded, shuffling the data can avoid non-stationarity and possible biases. **Research Agenda:** If data is accessed through a data service such as a database, this non-stationarity problem can be alleviated by providing a mechanism to read it by chunk in random order. OLA-RAW [50] (see Section 3.5.1) shuffles the data incrementally using more sophisticated mechanisms to avoid global pre-processing. While most databases offer data sorting (at a high cost), very few offer shuffling. Although the problem has been explored in research [237], existing systems offer few solutions.

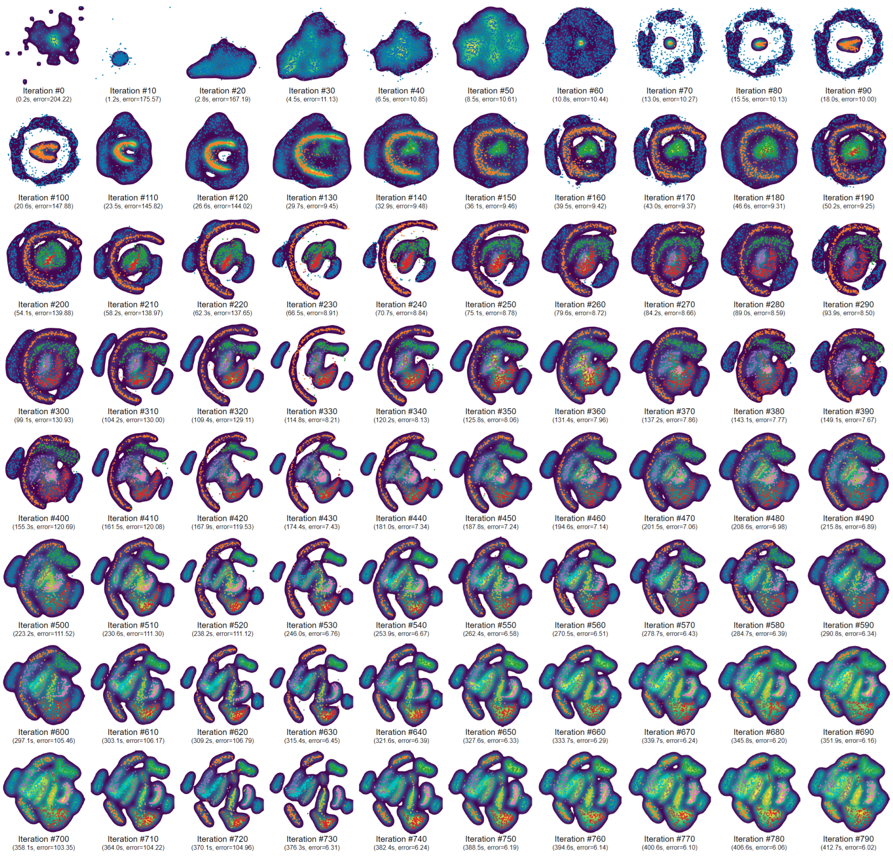


Figure 4.2: Responsive *t*-SNE [142] progressive steps for computing a multidimensional projection of the MNIST dataset loaded in chunks in a very skewed order (worst case). The algorithm eventually converges but is slower to deliver a faithful picture of the full dataset than if the input data were shuffled.

In many situations, when data is stored in one location and needs to be used (e.g., visualized) in another, there is no need to transfer all the detailed data, since it can be sent in an aggregated form. Aggregated values are typically histograms or single values that can also be computed progressively. Current databases only allow computation of the final exact results, but a progressive system could easily receive a progressive histogram or a progressive value with confidence intervals, since this is already what it deals with internally. This kind of service is not yet available, but is easily foreseeable and would be useful, e.g., for direct consumption of aggregated values on large data over the web. This is what progressive databases, as described in Chapter 3, aim to provide.

4.6 Specific Progressive Services and Strategies

In addition to fundamental algorithms and data structures, progressive computations need to have *strategies* to manage progressive updates and keep track of data changes in the computation pipeline. The ProgressiVis toolkit [83] explores some possible approaches. Although some of these are discussed here, other possibilities surely exist. **Research Agenda:** More research and engineering work is needed to better understand what strategies might be effective in managing interactive progressive computations in different exploration scenarios.

4.6.1 Progressive Stability and Normalization

As we will discuss in Section 5.2.1, a *stable* output or visualization is one that does not change or move excessively between progressive steps. In a chart, for example, the axes and color scales should remain consistent, changing as little as possible. There will often be a trade-off, then, when underlying data has changed substantially—should the visualization stay true to the previous steps or to the current state?

A useful instance of illustrating the trade-offs around stability comes from axis normalization. In many data analysis pipelines, data should be normalized: the values of a vector should be scaled to fit between e.g., $[0,1]$ or with a mean of 0 and a standard deviation of 1. The normalization process needs to know the exact bounds or distribution of the data before normalizing them. This is typically a two-pass operation. In a progressive setting, each new incoming chunk of data can change the bounds or the distribution, and if the normalization has to adapt at each chunk, then all of the normalized data may also change at each chunk. This may, in turn, affect other operations down the pipeline and may even trigger a change in the scale of the visualized data, producing an undesirable instability on the display. Different strategies can be devised to avoid these frequent jumps and resets in the computations and visualizations, such as temporarily extending the distribution or temporarily filtering out out-of-distribution data. Strategies are always required for visualization at the end of the pipeline. But in some cases statistical or machine learning operations may also need them too.

The first strategy—temporarily extending the distribution—consists of adjusting the bounds of the distribution when it has to grow by some percentage of the growth rate. For example, if the first chunk of data is bounded by $[-1,1]$, the maintained bounds will be constrained to grow by no more than 5% to become $[-1.05,1.05]$, ensuring that the bounds don't shift excessively. Extending the distribution with a fixed percentage of the growth stabilizes the rescaling process somewhat at the cost of keeping bounds that are too narrow, with a bounded error (5% in the example). This option can still suffer from sudden changes if extreme values are not well distributed in the data, but for shuffled data this simple strategy is a clear improvement over the baseline.

Instead of increasing the bounds for normalization, a progressive pipeline can filter out entries that are out-of-bounds until too many of them have been seen and then trigger a full recomputation. While interacting with the computation, the analyst can also monitor the out-of-bounds entries and trigger the recomputation only when desired.

Research Agenda: There are certainly other methods for dealing with progressive normalization, but having the user in the loop allows for simple yet effective solutions instead of more complicated ones, e.g., trying to model the true distribution from an incomplete one.

4.6.2 Data Change Management

When a progressive application is built from scratch and all the dependencies are well known, a centralized approach can be used to orchestrate the multiple algorithms and feed the data to all the concerned parts of the software as it arrives. In contrast, to implement modular progressive algorithms and data structures, one important property of a toolkit is to maintain a separation of concern between all its components. For this purpose, the ProgressiVis toolkit [83] provides a mechanism to allow progressive *computing modules* (the equivalent to progressive functions) to keep track of changes between each run in the data structures they operate on.

This issue can be related to the need for communication mechanisms between back-ends and the User Interface. Popular mechanisms such as the "Model-View-Controller" [164], the "Observer" design pattern [96, p. 293], and Functional Reactive Programming [323] all maintain a back-end to track what has changed, and a front-end that visualizes the differences. In interactive systems, changes happen every time the user moves the pointer or touches a key; in databases, changes occur with each commit. In the ProgressiVis system, changes come each time a progressive module completes a chunk or a certain number of iterations.

For example, returning to the normalization example above, the minimum and maximum values of each table column have to be maintained. A progressive module is in charge of computing and maintaining the minimum values of each column contained in a data table (call it the *Min* module), and another one for the maximum values. The Min module will be run each time the data table is modified, which is usually by chunk when the table is loaded or filled. So, if the data table is loaded from a CSV file, at every chunk (e.g., every 10,000 records), *Min* will run to update the minimum values it maintains. Therefore, at each run, it requests the changes from the last call in terms of a *delta*.

Similar to database triggers, the *delta* structure is made of three parts: the rows *created*, *deleted*, and *updated* (ignoring here that data columns can also be created, deleted, and updated).

- Assuming the progressive system is simply loading a large table, the *Min* module receives a delta with only created rows. Its task is simple: to compute the minimum over the new row values and combine it with the already computed running column minima.
- If the progressive system is computing the Min over a table filtered by a user, using e.g., a range slider, ranges of values will be filtered out or added back. The delta will thus contain deleted and created values, and things get more complicated. The simplest strategy is to restart the module to compute the minimum values from scratch. There are potentially smarter strategies that can be used, but they are also more complex. For example, one of the ProgressiVis toolkit Min modules is optimized to handle deleted values; it keeps track of the indices that reached the running minimum value of each column. If the deleted values do not intersect with these indices, there is no need to restart from scratch since the minimum has not been changed.
- If the progressive system is computing the content of the table iteratively, e.g., the x, y columns computing by a graph layout algorithm, the computed values

will change with each iteration and the Min module will receive deltas with *updated* items. In that case, resetting the module might be the easiest and most efficient approach.

The communication mechanism based on deltas allows non-centralized handling of the data changes in algorithms/modules, but it adds some burden to each module.

Different algorithms handle these deltas in different ways. Some algorithms can ignore them (e.g., many iterative algorithms such as k-means clustering), while others can only deal with creations and in other cases need to restart from scratch. The level of complexity for managing these deltas varies across algorithms, but the information they carry is, in most cases, essential.

This *delta* service to keep track of what happened to a data structure since the last time it was processed is very useful to progressive libraries and algorithms in maintaining good separation of concern between algorithms and state management, but it requires some specific data management efforts from the data structures. The implementation of that mechanism is slightly different from database triggers. Database triggers can be heavy to implement since they require the database to keep track of all the changed values of all the tracked tables to retrieve the old values of updated and deleted rows in the trigger management procedure (though not all databases provide these previous values). ProgressiVis management is lighter because it does not keep track of previous values, but only the indices that have changed, which it does through a very efficient data structure based on compressed bitsets [47]. This mechanism of change tracking is implemented in several progressive-aware data structures available in ProgressiVis: tables, columns (that also implement vectors), dictionaries, and bitsets.

Research Agenda: This state management is specific to ProgressiVis, and there may be alternative ways of maintaining the separation of concerns between progressive algorithms and data structures, but some mechanism should be provided to keep track of the changes to the data structures that occur when re-entering progressive algorithms. This problem is novel and calls for more research.

4.7 Summary

Like every programming environment used in data analysis, PDA will need to provide a set of algorithms and data structures suited to progressive computations. These data structures and algorithms have to provide a particular API to be usable progressively: the functions should not only return values, but also some quality information that is useful in assessing their utility, and the data structures should be able to compute what changed since the last time they were used by a progressive algorithm. Progressive systems should also provide a variety of algorithms that are suitable in different situations: *progressive-in-chunks* or *progressive-in-iterations*.

Some implementations will accommodate both of these cases, but will have performance issues. In addition, progressive algorithms should be used with particular strategies to avoid triggering complete recomputations, and these strategies are new and need to be studied further. Finally, on top of these requirements, progressive libraries or toolkits should also implement internal mechanisms to track changes in data structures that are used multiple times by the algorithms. The ProgressiVis toolkit implements a method based on deltas over its data structures so that each algorithm can determine what changed since the last time they ran. There may be other possible mechanisms, and they need to be studied and compared for efficiency and usability from a programming point of view.

Overall, the development of fundamental algorithms and data structures for progressive systems will require a new line of research for adapting existing algorithms. But it will also be necessary to devise software engineering patterns and architectures to facilitate the design and development of progressive systems.

The mean and median examples in Section 4.2 represent an easy and hard case of algorithms regarding progressivity. Wickham [330] distinguishes three types of summary statistics: distributive (e.g., count, sum), algebraic (e.g., mean, standard deviation), and holistic (e.g., median). **Research Agenda:** The first two are easy to compute progressively, but the last one requires more research on approximate methods, including data sketching.

Research Agenda: More work is needed to characterize existing algorithms in terms of progressivity and to help in the discovery of generic transformation methods or strategies to make more of them progressive.

The adaptation of algorithms to become progressive can be viewed as an extension of transforming algorithms to become online, with soft real-time constraints. It can also be viewed from the perspective of streaming algorithms to allow them to adapt to changing data, whereas data sketches are only implemented for data streaming in and not updated over time. Several algorithms have already been transformed into progressive counterparts, and we look forward to the development of more of these adapted or new algorithms.

5 Visualization

5.1 From Classic to Scalable Visualization 70
5.2 From Scalable to Progressive Visualization 73
5.2.1 Progressive Rendering 73
5.2.2 Interaction, Steering, and Control 78
5.3 Designing a Progressive Visual Analytics System 79
5.3.1 Choose the Right Visual Mapping 79
5.3.2 Represent the Changes 80
5.3.3 Visualizing the Progression 81
5.3.4 Interaction Between Visualization and Progression 82
5.4 Existing Visualization Approaches in PDA 82
5.4.1 DimXplorer 83
5.4.2 InsightsFeed 85
5.4.3 ProReveal 87
5.5 Challenges for Progressive Visual Analytics 89
5.6 Summary 90

Authors Marco Angelini and Jaemin Jo

User interface designers for PDA systems face a unique challenge: they not only have to enable the user to interpret the final results of a computation, but also to make sense of intermediate results, assess uncertainty, comprehend the progress of progressive computation, and steer the computation. A poorly designed visualization can lead the user to incorrect findings or imply unfounded accuracy, reinforcing her biases. Well-designed visualizations and user interfaces are therefore critical in PDA. We refer to the visualization aspects of PDA as Progressive Visual Analytics (PVA).

In this chapter, we begin by discussing the visualization community's journey from classic visualization to progressive visualization. Starting with classic visualization techniques, typically for small-scale data, we offer an overview of the history of the attempts that have been made to allow visualizations to scale to the large data sizes that are common in PDA. We then raise some of the questions and challenges that come up when visualization is used for a sequence of progressively changing data in

PVA. To illustrate how these challenges can be met, we briefly describe three PVA systems that grapple with these questions and conclude with a future research agenda for PVA.

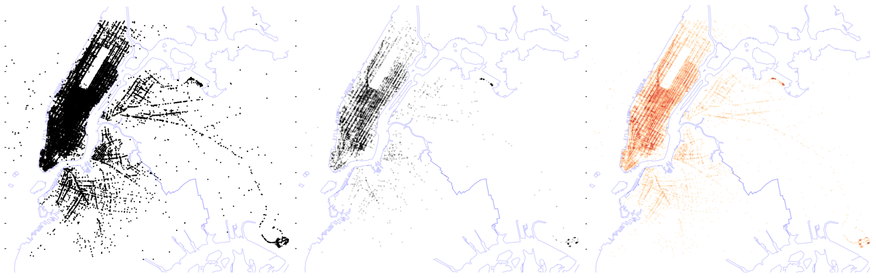
5.1 From Classic to Scalable Visualization

Although the scalability of visualization is a major concern even in classic visualization systems [249, 330], its importance becomes more evident in PVA, especially with *progressive-in-chunks* involving very large datasets. As shown in Figure 5.1, many common visualization techniques, such as scatterplots, parallel coordinates, and pie charts, do not scale well graphically. In particular, techniques where each data item is mapped to a single graphical element are subject to overplotting, as the number of visual elements in the scene is bound to the data size, which can grow arbitrarily large.

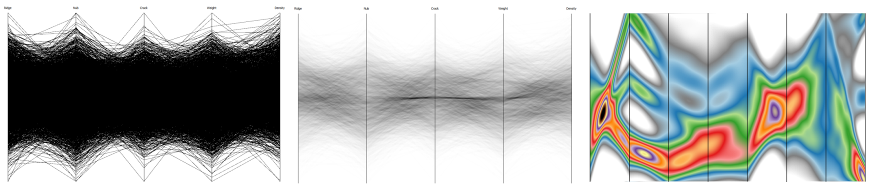
Figure 5.1 highlights some of the main issues related to visual scalability: *number of items*, *number of dimensions*, and *number of distinct categorical values*. One prominent issue with non-scalable visualization techniques in Figure 5.1 is the overplotting problem, which results from the number of visual items drawn. There are five methods typically used to address visual scalability: *sampling, filtering, aggregation, dimensionality reduction*, and *summarization*.

The most straightforward solution to the problem of the number of items is to reduce the number of elements that have to be shown. *Sampling* achieves this, but it should be used with care, since important but infrequent patterns in data, such as outliers, can be left out [25] (see Figure 5.1a (center)). *Filtering* is another method for reducing the number of items and can be effective if the selectivity of the filter applied is high enough. However, both methods cannot support overview tasks where the full dataset should be visualized.

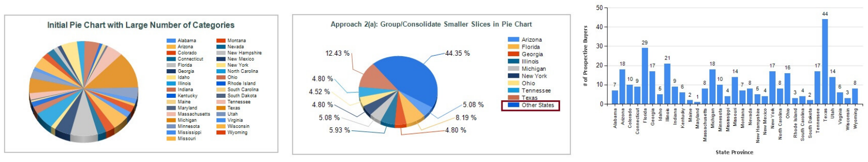
To address the overplotting problem while faithfully showing all of the data, one can map multiple data items to one visual element, as in *aggregation*-based techniques [330]. For example, splatterplots [190] overcome the overplotting problem of conventional scatterplots by representing dense data points as contours while showing individual outliers as points around the contours. Density maps are another approach to scaling scatterplots. These maps split a 2D data space into 2D bins, count the number of items belonging to each bin, and represent the counts, typically using color (see Figure 5.1a (right) and Figure 5.1b (right)). One benefit of density maps is that the number of visual elements drawn (i.e., the number of 2D bins) can



(a) Visualizing locations of 100k yellow taxi pickups in NYC [214] (sample from May 2016) produces overplotting in the center of Manhattan (left). Sampling (center) avoids overplotting but also erases information, such as the multiple pickups occurring along the paths from the airports to Manhattan. Using a density plot (right) with appropriate parameters allows for showing the main patterns and preserving sparse areas.



(b) Parallel Coordinates on large datasets also produce overplotting (left). Transparency (center) can alleviate the problem for a few thousand items. Density-based parallel coordinates (right) partially mitigate this on large datasets, showing the main trends but hiding outliers. From [115].



(c) Pie charts do not scale well with the number of categories in general (left). Gathering small categories in an "other" category (center) can sometimes alleviate the problem, but using a more appropriate visualization, such as a bar chart, can be a better option (right). Courtesy of `https://www.mssqltips.com/sqlservertip/2773/handling-a-large-number-of-categories-in-a-sql-server-reporting-services-pie-chart/`

Figure 5.1: Standard visual representations do not scale well as data grows. Scatterplots suffer from overplotting, as well as parallel coordinates. Even pie charts become unusable when the number of categories grows. All of these techniques require expensive analytical methods to correctly handle scale.

be user-defined, completely independent of the data size, which allows the rendering process to finish within a predictable time.

Unfortunately, not all non-scalable visualization techniques have scalable counterparts. For example, bar charts, a very basic visualization technique, scale well for the count of items represented in bars, but not for the number of bars, the number of distinct categories usually represented on the x-axis. It is difficult to address this problem using simple aggregation, as in Figure 5.1c (center). Furthermore, adopting aggregation-based techniques can present new design challenges for visualizing the information that could be easily shown in the original non-aggregate plots. For example, in a conventional scatterplot, the class membership of an item can be easily encoded as the color of the corresponding point. In density maps, because multiple items from different classes can be aggregated into the same 2D bin, showing the class membership introduces new considerations, as discussed in Jo *et al.* [143].

Parallel Coordinates and Scatterplot Matrices are limited to visualizing tens of dimensions. When the number of dimensions increases, visualization can use *dimensionality reduction* methods, which are either steered by human selection, i.e., showing the bivariate relationships between the chosen attributes as a SPLOM, or based on automatic techniques such as t-SNE [311] or UMAP [192], i.e., showing a low-dimensional projection of high-dimensional points as a scatterplot. The reduced dimensions can help place items in space with a meaningful distance function, at the cost of losing interpretable dimensions. These algorithms are expensive to compute, but progressive implementations do exist [142, 160, 232, 233] (see Figure 4.2).

Another possible solution to scalability consists of using *summarization* techniques, such as presenting distributions of the data. For example, the scagnostics [333] technique computes a number of descriptive techniques for different dimensions, such as how skewed and how sparse the data is, akin to a dataset "facial composite".

Most of the known techniques used to scale visualizations are expensive to compute. Even sampling, which is fast for very simple techniques, can become expensive when trying to avoid artifacts [25, 48, 317]. Therefore, visualizing data at scale implies an extra cost not only because of the rendering, but also because of the analytical cost of controlling visualization quality and achieving a meaningful rendering.

Furthermore, to remain interactive, visualizations should react with low latency, which is something that eager scalable visualizations cannot guarantee. One important advantage of PDA is that it allows scalable visualizations that are as interactive as their small-data equivalents, provided they are properly adapted.

5.2 From Scalable to Progressive Visualization

PVA introduces four new groups of issues: (1) progressive rendering; (2) interaction, steering, and control in a progressive setting; (3) PDA uncertainty, as addressed in Chapter 6; and (4) new tasks such as monitoring, attention management, and early decision-making, as addressed in Chapter 7.

5.2.1 Progressive Rendering

In a classic visualization, where data are fully available at the time of rendering, we expect to be able to create the visualization in a single step. By contrast, a progressive visualization is updated at a rate that depends on the progress of intermediate results. The visualization and graph drawing communities have long been aware of the problem of representing a series of partial results from a progressive computation [54, 117, 216, 290]. Graph drawing with spring models dates back as far as the early 1980s [74].

In the last 20 years, there has been a greater focus on big data analysis coupled with a rich data visualization environment and the rise of Visual Analytics [154] as a discipline. The first works on the topic of instrumenting, steering, and evaluating PDA with the use of visualization appeared with Williams & Munzner [334], Fisher *et al.* [89], and Angelini *et al.* [9]. The field was labeled Progressive Visual Analytics by Stolper *et al.* [293].

As a pioneering progressive visualization, Fisher *et al.* [89] chose to use a histogram in their SampleAction system for its simplicity and familiarity (see Figure 5.2): the height of a bar shows the estimated value from each intermediate partial result, while the error bars show the confidence bounds around the relative estimate. The error bars show the range of values that may occur at the confidence levels. In this way, the user can issue a query and inspect the partial results visually by interpreting column estimates and the probability of them being included in a shrinking confidence interval. The progressive histogram in the SampleAction system illustrates well the new design concerns that arise in the transition from scalable to progressive visualization. The two prominent concerns are the **stability** and **uncertainty** of progressive visualization.

Stability A progressive visualization needs to be updated to faithfully show the most up-to-date results. However, updating a progressive visualization generally means confronting a tension between quality and stability, especially with aggregation-based

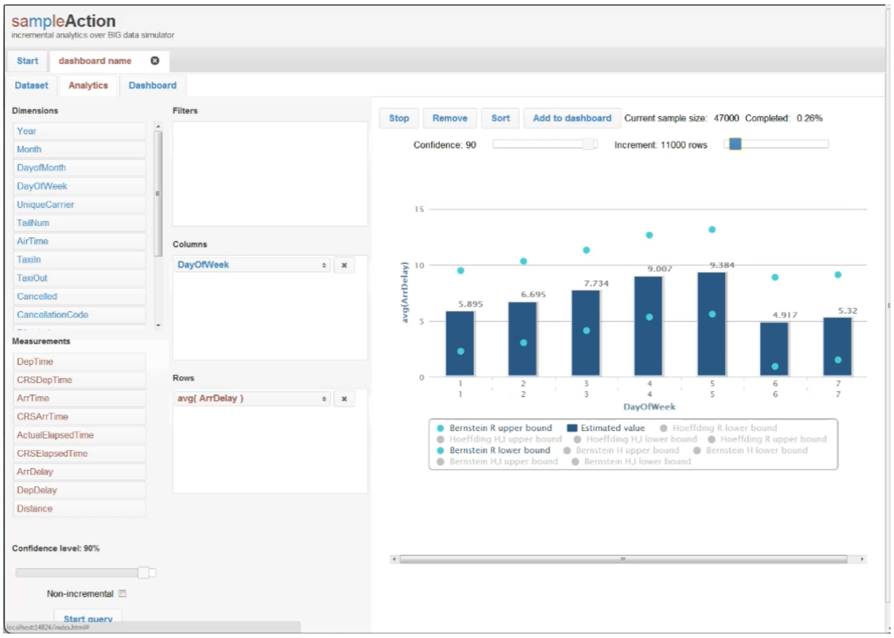


Figure 5.2: A screenshot from the SampleAction system [89]: the analyst is looking at flight delays by day of the week. The histogram on the right represents the current estimates with dark blue bars, and the expected range of values with pale blue dots. A progress indicator on top shows that just 0.32% of the dataset has been processed.

techniques.

In classic visualization, only one state exists for the visualization: the final one. But in PVA, the system must produce a visualization for each intermediate result. This consideration has consequences even for simple visual mappings.

For example, consider a bar chart showing the top 10 categories and with values in descending order. As more data points arrive progressively, some categories may enter or leave the top 10, the order might change, and the axes may need rescaling. Not unlike the discussion about tracking normalization in Section 4.6.1, the visualization designer must choose whether to maintain the precision of the latest data or stability.

Stability refers to the need to preserve, to the fullest extent possible, all of the

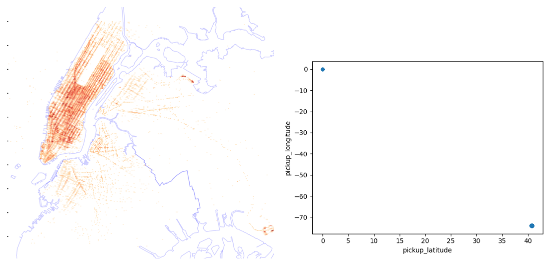
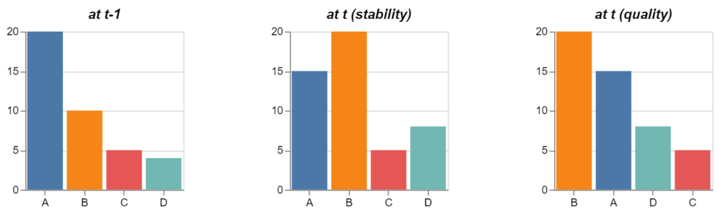


Figure 5.3: In the NYC Taxi dataset [214], the density map of taxis loaded progressively (left) becomes the top left point (right) after loading a data row representing a taxi that traveled to Florida with its meter on, creating a scale problem.

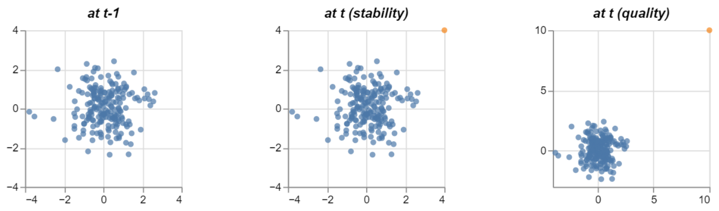
aspects of the visualization that might help the user maintain their mental model. These include the scale and order of the axes, the color mapping, and all other aspects of the visual mapping. In general, keeping these stable will allow the user to more easily understand the changes that are made.

The nature of progressivity highlights some of the challenges of stability. Outliers distorting an axis are an excellent example: in the NY Taxi dataset [214] shown in Figure 5.1a, one taxi forgot to turn off its meter when traveling to Florida. When its record arrives in a chunk, the scale dramatically changes and the NYC area shrinks to a few pixels, as shown above.

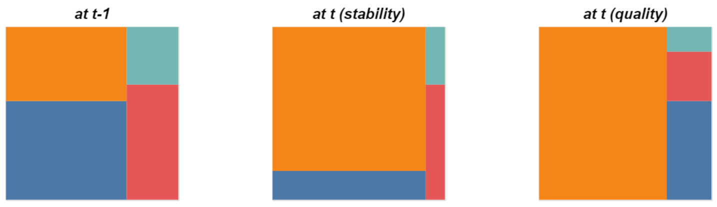
We have discussed some approaches to improving stability in Section 4.6.1. Indeed, many algorithms can be stabilized at the cost of layout quality. A progressive Treemap is an interesting example (Figure 5.4c). One can fully re-layout and render the Treemap when new data arrives, independently of the Treemap that was previously on the screen. In this case, a layout can be obtained that is optimized to the current data in terms of quality metrics such as aspect ratio [289]. However, since the coherence between the old and new Treemaps is broken, the user will see a sequence of Treemaps that “dance” with each chunk, which makes it harder for the user to visually track the changes and understand temporal patterns. On the other hand, one can consider the previous layout when computing the new one, seeking better stability between them. In the Treemap example, one can minimize the relative position change for the same rectangle between the frames [289, 315] at the expense of visual quality. There are similar tradeoffs for graph drawing: a stabilized transition may not choose as optimal a layout [202], as a drawing from scratch would.



(a) It is common to sort the bars in a bar chart by their height (left). When the heights change due to progressive computation, maintaining the order of the items on the x-axis may conflict with maintaining the ordering on the y-axis (center and right).



(b) To make scatterplots more stable, the domains of the axes can be fixed, which means that an out-of-range outlier that is found during progressive computation (left and center) will not be shown. When the domains are updated to reflect the change, the visualization becomes less stable (left and right).



(c) When a Treemap is updated, one can maintain the topology of the rectangles in the previous frame for better stability at the cost of visual quality, i.e., aspect ratio (left and center). By contrast, fully rerendering the Treemap loses topological consistency (left and right).

Figure 5.4: When a visualization is updated progressively, there is often a tradeoff between stability and quality.

Finally, it is worth contrasting progressive computation with progressive disclosure. While in the former, the progressive nature comes from the computation side (e.g., new data chunks computed), in the latter, all the data has already been processed and the progressiveness is applied only on the rendering part to ease comprehension or provide information about the provenance of the result and relative visualization. For example, Rosenbaum *et al.* [252, 254] explain how it can be valuable to progressively show a user more of a pre-computed treemap or parallel coordinates view, depending on their screen size. Interestingly, these visual mappings do not update the layout of the visualization, mitigating the tension between stability and quality. On the other hand, they require the full pre-computation of the data and visual mapping, limiting their application to cases in which this is reasonable in terms of the time and resources needed.

One possible solution to address this trade-off between stability and quality is to maintain stability by default and allow the user to interactively trigger "repairing" of the visualization, explicitly breaking the stability to achieve better quality, as mentioned by Jo *et al.* [142]. **Research Agenda:** This strategy has implications for the whole progressive system, since algorithms should inform the user that repair actions are available. Changing the visualization layout certainly breaks the user's mental map and would require mechanisms to help users recover it, e.g., through history navigation or animated transitions.

Uncertainty and Quality In contrast to classic visualization, progressive visualization can exhibit a high level of uncertainty when the computation is not yet complete (see Chapter 6). The visualization needs to communicate this *PDA uncertainty* using extra visual channels. For example, the SampleAction system superimposes whiskers on each bar to convey the error information as a *confidence interval*. Pangloss similarly shows a pair of heatmaps, one showing the aggregated scatterplot and a second showing the uncertainty of that same scatterplot [205]. Patil *et al.* [224] study the effectiveness of techniques to convey the uncertainty of progressive bar charts and conclude that humans can interpret some of them well. In some systems, it is impossible to bound the approximation, but it is still valuable to present the user with a level of *quality* to illustrate how far the algorithm has advanced.

These are all confounded with other sources of uncertainty in visualization. Visual encodings themselves may try to convey uncertainty (e.g., "value suppressing uncertainty palettes" [60]), and the scaling issues mentioned in Section 5.1 can cause additional uncertainty, as users become unable to read an overplotted or heavily sampled plot. Users consuming the analysis can be prone to perception and interpretation bias, which can be exacerbated during PDA, as discussed in Section 7.5.

The next chapter (Chapter 6) further discusses the issue of identifying, quantifying, and visualizing uncertainty in PDA. **Research Agenda:** While there is already a body of research on visualizing quantitative uncertainty, visualizing progressive uncertainty is a newer area that calls for further study.

5.2.2 Interaction, Steering, and Control

In addition to the visual considerations, Progressive Visual Analytics introduces new issues related to interaction. These include interacting with visualizations (R12), steering analytical processes through their visualization (R13, R14), and controlling the progressive process itself (R15, R16).

Interaction with progressive visualizations The dynamics of PVA completely change how we define and realize user interaction. To illustrate these issues, consider a situation where a user makes a selection by brushing on a scatterplot. With a static scatterplot, the intention of the user is clear: the user wants to select the items in the brushed area. But with a progressive scatterplot, things become more ambiguous. Does the user want to select the points by their current values? If some points in the scatterplot change values and now move outside the range, should they be removed from the selection as well? Are the points added progressively to the scatterplot also added to the selection automatically? If the axes of the scatterplot change, how should the range of the brush selection change?

In addition, points on a progressive visualization can have uncertainty associated with them. Points have a probability of being inside a certain range. What are the semantics of selection with uncertain data?

These issues can be illuminated by understanding the underlying goal of the user. For example, if the user wants to select some outliers on a changing visualization, does the system need to notify the user if the selected points are no longer outliers? The system should clearly inform the user how the meaning and outcome of interactions are being managed throughout the progressive computation. Examples of work in this area include Falcon by Moritz *et al.* [206] for managing uncertainty and level of approximation while interacting with data, Turkay *et al.* [304] where a specific adaptation of brushing technique for progressive systems, called "keyframed brushing", is proposed, Badam *et al.* [16] where the visualizations are frozen while interaction is in progress (consistent with R13), and Högrafer and Angelini *et al.* [123], where an adaptation of brushing expressing a user's query on a bi-dimensional scatterplot allows steering of the progressive visualization and quickly prioritizing the generation and visualization of all the data items answering the query. **Research Agenda:** A number of questions remain to be explored concerning how to make use of the user's interactions with direct visualization.

Steering Progressive steering is another challenging area. The idea of steering (see Section 2.2) refers to the ability of the user to give direction back to the computation. Steering includes query steering, such as interactive filtering of data to remove irrelevant results, and hyperparameter steering in machine learning, such as interactively changing the number of clusters in a progressive *k*-means algorithm [16]. Some kinds of steering can produce immediate feedback, but others might need more time to become noticeable, while users need immediate feedback, at least to confirm that the changes requested are being taken into account and will be visible soon. **Research Agenda:** Some hyperparameters might be hard to steer due to the computation cost of changing them, or might require recomputation from the start, while others can be adaptively adjusted. A progressive system may warn users when they try to change them, but more research is needed to allow exploration of the hyperparameter space using PVA.

Control PDA design requirements listed in Table 2.1 mention several types of controls that PDA systems should provide (R12–R16), such as pausing or canceling the computation. Others are listed in later publications, such as navigating the history of visualizations in [16] and slowing down a progressive process to better understand it [314]. Jo *et al.* [142] also mention the algorithm notifying the user that triggering a recomputation will improve the visualization at the cost of reducing stability, and so potentially breaking the user's mental map. **Research Agenda:** Further work should be done to identify the types of controls that are needed by PVA and evaluate their pro and cons on real systems.

5.3 Designing a Progressive Visual Analytics System

The process of adding a visualization for PDA poses a set of constraints and trade offs that need to be managed during the design process.

5.3.1 Choose the Right Visual Mapping

The first important step is the choice of a visual mapping that is able to support the PDA computation. While, in principle, this step is also applicable to data visualization more generally, it must take into consideration how well the visual mapping chosen fits in terms of the production of partial results coming from the PDA process and their effects on the visual channels used (R1 and R2).

As we discuss in Section 5.2.1, the requirements for a chart with data that can change dynamically will have different implications. The mapping must be chosen to ensure that the visual mapping will be scalable (in the case of big data visualizations) as well as robust to stability challenges.

Several papers propose design requirements for a progressive system, touching on visualization aspects [11, 16, 207, 304]. **Research Agenda:** We need guidelines on how to meet visualization requirements, how to select the best visual encoding for a progressive visualization, and which features it should provide.

5.3.2 Represent the Changes

The second design constraint concerns the representation of changes resulting from the PDA computation. When the progressive computation is completed, the visualization is updated and must choose when and how these updates should be shown. We can summarize these as *when* and *how* to visualize the changes.

When to visualize changes? The simplest strategy would be to always visualize, using the chosen visual mapping(s), all the changes (or partial results) produced by the progressive process. However, this may not necessarily result in a smooth visual convergence toward a visually stable result, but rather in a sequence of abruptly changing visualizations that can potentially distract the user (against R4) and diminish their trust in the visualization. An unstable visualization will be challenging for a user to interpret, as explained in Section 5.2.1. A rapidly changing visualization is likewise challenging to interact with.

Consider the confusion generated by an unstable visualization. A frequently changing layout makes it hard to track single elements, meaning that the user is exposed to confusing rearrangements of elements in position and size. Worse, any single partial result could be an artifact or an outlier; it may take multiple partial results to recognize a converging trend. Smoothing both the visual mapping and the results can help keep the user from being confused. For example, in the work by Angelini and Santucci [9], visualizations of different partial results that are considered very similar are omitted from rendering so as not to disturb the user with insignificant visual changes, while speeding up the progression. In Badam *et al.* [16], the user can choose between a monitoring mode where the visualization is automatically updated and an interaction mode where the visualization is only updated when requested. This implies that the visualization has to be notified by the underlying algorithm(s) that refreshing will reveal new information (R13).

How to visualize changes? The changes from the progressive computation should be conveyed to users based on the visual mapping being used. A recent survey by Ulmer *et al.* [307] investigated the main aspects of progressive visualization. Among these, the authors identified the *visual update pattern* as a prominent aspect and one that is dependent on the chosen visual mappings. Visual update pattern types describe how visualizations are progressively updated when new data chunks or algorithm iterations are produced. This pertains to factors like animation design and controlled update frequency, which are directly relevant to what has been discussed about the question of when to visualize changes. Moreover, it also concerns visual attribute mapping and the pattern of updates, for which the authors identify two main types: *extension* and *overwrite*.
Extension denotes the process in which new visual elements are added to the visualization, which is extended to display new data. Old data is not lost and can help the user preserve context while new data is coming in.
Overwrite, on the other hand, removes old results and presents just the new visualization state. The authors model it as a partial overwrite, where only the elements that were changed with new partial results are modified, and a full overwrite, where the previous results are removed completely and new results are displayed. There are pros and cons with both strategies, and the authors investigated their spread with respect to the main visual mappings: temporal, geospatial, hierarchical, and multidimensional visualizations seem to present an even representation of both strategies, while network and field visualizations show a predominance of the overwrite strategy.

Finally, an important aspect is the visual communication of the changes that are taking place. Since the progressive visualization needs to be updated, it may be helpful to highlight to the user where changes are happening. In aggregate visualizations, visual glyphs may change size, shape, as well as location.
Research Agenda: There are opportunities for design languages to show changes in progressive visualization.

5.3.3 Visualizing the Progression

The third important aspect to consider is how to visualize the data progression coming from the progressive process itself (R10, R11). In PDA, there is a second layer of data, referring to the progressive process itself. The system must provide elements that describe how the progression is unfolding (e.g., progress bars that show the progression of computation) and allow the user to infer the quality of the visualized results. The way in which these quality indicators are visualized and communicated to the user is an important consideration during the design of visualization for PDA.

A line of previous works points to a clear division between properties that concern the progression and those that concern the application domain data, with the latter simply reacting to the changes produced in the former. Several projects represent changes from the progressive computation directly on the main visualization [83, 293, 304]. This saves visual space, allowing the visualization to dedicate more space to the visual representation. This technique can be most effective when both the analysis technique and the representation both refer to a clear concept of uncertainty.

Others complement the visualization with an area dedicated to metrics governing the progression itself [16, 89, 270]. At the cost of both using more space and dividing the user's attention, these techniques allow the system to portray uncertainty or progress in greater detail. For example, in Fisher *et al.* [89], the visualization shows two density maps: one for the estimated outcome and another for uncertainty in the data. A different approach for that work might have instead used a value-suppressing uncertainty palette [60].

Research Agenda: More research is needed on the advantages, disadvantages, and design patterns for this topic.

5.3.4 Interaction Between Visualization and Progression

In progressive visualization, the user should be able to control the progression (R12, R13, R14, R15, R16), thus avoiding new design requirements. We will discuss different roles for interaction in Chapter 7. The system should allow the user to interact with the progression, for example, by pausing and resuming computation. Some systems may allow the user to intervene in the computation, for example, by changing parameters. With slow visualization and computation, other systems may allow the user to schedule multiple ongoing visualizations with limited resources.

5.4 Existing Visualization Approaches in PDA

In this section, we briefly describe three research PVA systems to demonstrate how the aforementioned considerations are realized in practice. Each one exemplifies a combination of design choices, and we analyze how each example responds to the considerations we present. These examples do not cover the full range of design possibilities, of course. **Research Agenda:** A systematic approach is needed to identify the design space of choices for controlling and interacting with PVA. The recent survey on progressive visualization by Ulmer *et al.*is a promising first step in this direction.

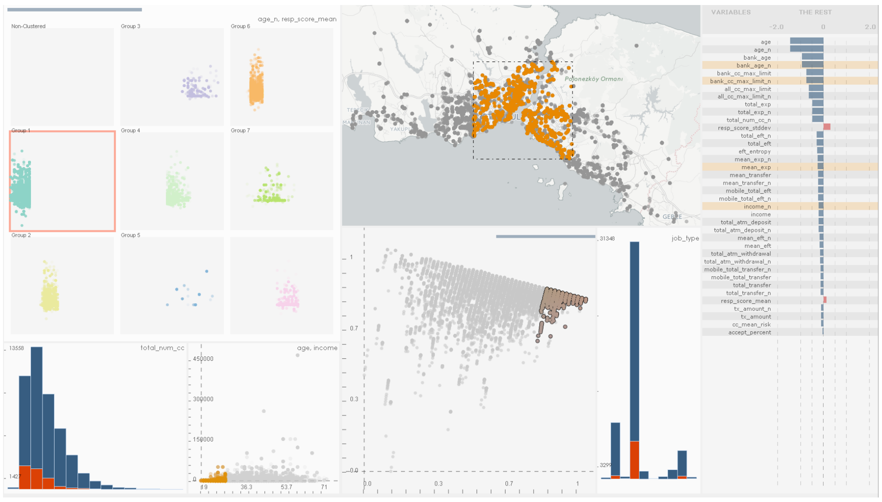
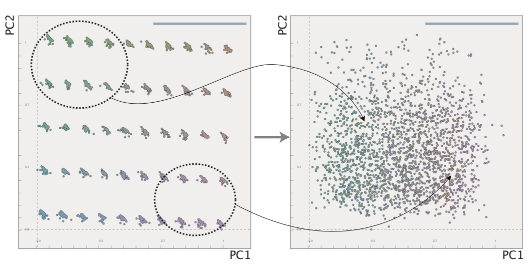


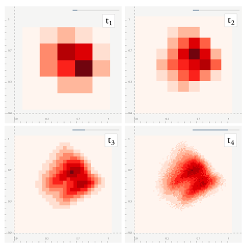
Figure 5.5: The DimXplorer System [304] adopts an adaptive sampling strategy to deliver progressive responses.

5.4.1 DimXplorer

DimXplorer [304] (Figure 5.5) is an interactive and progressive analysis system for exploring high-dimensional data. This system introduces two analysis techniques into progressive data analysis: Principal Component Analysis (PCA) and *k*-means. To guarantee the response time, the system employs the two techniques using online algorithms, Online PCA [255] and Mini-batch *k*-means [272]. These online algorithms work on small pieces of data or *batches*, generating and delivering intermediate results quickly during the computation. These intermediate results become more



(a) Color-based transition



(b) Density maps with different bin sizes

Figure 5.6: Visual components in DimXplorer [304]. (a) Assigning unique hues to points and maintaining them during transition can help the user track the changes. (b) Bin sizes are used as a visual cue for progression. The bins become narrower as the computation progresses, revealing more details.

accurate as more batches are processed and are used to update the visualizations in the interface.

The size of batches plays an important role in ensuring that the responsiveness of the system is below the progressive quantum, as explained in Section 4.7. If the batch size is too large, the system will not be able to achieve the desired response time, while a too-small batch size will undermine the system's throughput. DimXplorer adopts an adaptive sampling strategy to fine-tune the batch size. This iterative algorithm starts with an initial batch size, *b*, that is estimated to be small enough to deliver the response in time. If the actual execution time is longer than the desired time, *b* increases by one factor, and if it is not, *b* decreases by another factor. In this way an optimal batch size is found while observing temporal constraints.

To provide coherency between visualizations and maintain the pace of interaction, the system introduces the following strategies. The system seeks consistency between two consecutive computation results. For example, we know that PCA by nature finds principal components (PCs) with arbitrary rotations and signs; we can find mirrored and/or flipped PCs even for the same data. If we display the projection result as is, it would be extremely hard for humans to follow the change. The system addresses this problem by checking the Pearson correlations of the axes between the first and second PCs and mirroring and/or flipping PCs if needed.

DimXplorer supports animated transitions in visualization, providing temporal coherency for visual elements. An animation sequence takes one second, which is consistent with the desired response time of online algorithms. In addition to transitions, color is used to reveal the relationships between the two scatterplots. Each data point in a scatterplot is assigned a color based on its position, and the color is maintained during the transition (Figure 5.6a).

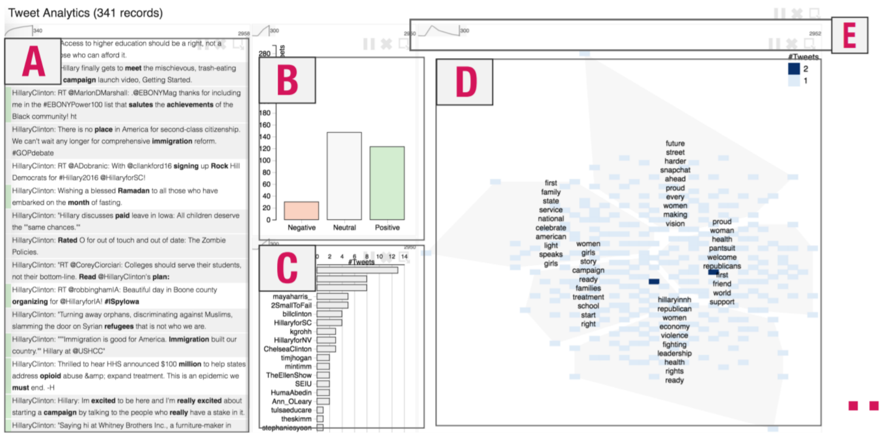
To provide animated transitions between clustering results, it is necessary to identify a mapping between two consecutive results, i.e., which cluster in the current result (C_{cur}) corresponds to which cluster in the previous result (C_{old})? To this end, the authors measured the similarity between each pair of clusters (one from C_{old} and the other from C_{cur}) by computing the Jaccard coefficients and setting the cluster with the highest coefficient as counterpart.

Although a one-to-one visual mapping (i.e., scatterplots) is the default visualization approach, the user can switch to a many-to-one visual mapping, or density maps, if necessary. One interesting idea is for the authors to adaptively adjust the bin size of the density map to reflect the progress of the computation—the bins become narrower as the computation progresses—as a means of representing the uncertainty (Figure 5.6b). In this approach, the interpretation of fine details in early visualization results is deliberately made more difficult, thereby preventing the user from making decisions too quickly, while also providing extra clues regarding the progression and uncertainty, in addition to the progress bars.

5.4.2 InsightsFeed

InsightsFeed [16] is a progressive visual analysis tool for exploring Twitter data at scale. Its interface consists of three views (Figure 5.7), which are updated progressively: a tweet list view (a list of tweets), sentiment and popularity visualizations (bar charts), and a tweet map (a density map for a *t*-SNE projection [311] and *k*-means [186] clustering of tweets). Like DimXplorer, InsightsFeed adopts animated transitions to show updates in views. The difference is that it adopts staged animations to maintain cognitive workflow on updates. For example, the axes of a bar chart are updated first, then new labels are added, and finally, the length of the bars is adjusted (Implication 1).

Each view in InsightsFeed has a dedicated control panel (Figure 5.8) to communicate and control the progress (Implication 3 and Implication 5). This small widget allows the user to play, pause, and stop the progression (R15) and steer the hyperparameters of progressive algorithms (R14), such as chunk size (the number of tweets in a chunk), hyperparameters of the *t*-SNE algorithm, and the maximum

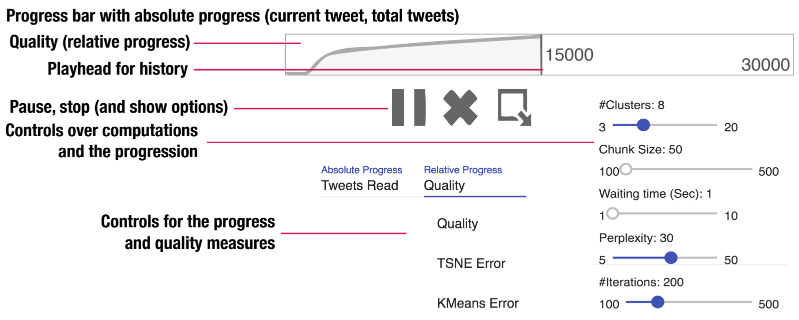

Tweet Analytics (341 records)
A
B
C
D
E
Negative
Neutral
Positive
#Tweets
mayaharris_
2SmallToFail
billclinton
HillaryforSC
kgrohh
HillaryforNV
ChelseaClinton
timjhogan
mintimm
TheEllenShow
SEIU
HumaAbedin
Ann_OLeary
tulsaeducare
theskimm
stephanieoyeon
first family state service national celebrate american light speaks girls
women girls story campaign ready families treatment school start right
future street harder snapchat ahead proud every women making vision
hillaryinnh republican women economy violence fighting leadership health rights ready
proud woman health pantsuit welcome republicans first friend world support

Access to higher education should be a right, not a
se who can afford it.
Hillary finally gets to **meet** the mischievous, trash-eating
campaign launch video, Getting Started.
HillaryClinton: RT @MarlonDMarshall: .@EBONYMag thanks for including me in the #EBONYPower100 list that **salutes** the **achievements** of the Black community! ht
HillaryClinton: There is no **place** in America for second-class citizenship. We can't wait any longer for comprehensive **immigration** reform. #GOPdebate
HillaryClinton: RT @ADobranic: With @cllankford16 **signing** up **Rock** Hill Democrats for #Hillary2016 @HillaryforSC!
HillaryClinton: Wishing a blessed **Ramadan** to all those who have embarked on the **month** of fasting.
HillaryClinton: "Hillary discusses **paid** leave in Iowa: All children deserve the ""same chances.""
HillaryClinton: **Rated** O for out of touch and out of date: The Zombie Policies.
HillaryClinton: "RT @CoreyCiorciari: Colleges should serve their students, not their bottom-line. **Read** @HillaryClinton's **plan:**
HillaryClinton: RT @robbinghamIA: Beautiful day in Boone county **organizing** for @HillaryforIA! **#ISpyIowa**
HillaryClinton: "Turning away orphans, discriminating against Muslims, slamming the door on Syrian **refugees** that is not who we are.
HillaryClinton: """Immigration is good for America. **Immigration** built our country."" Hillary at @USHCC"
HillaryClinton: Thrilled to hear HHS announced $100 **million** to help states address **opioid** abuse &amp; expand treatment. This is an epidemic we **must** end. -H
HillaryClinton: Hillary: Im **excited** to be here and I'm **really excited** about starting a **campaign** by talking to the people who **really** have a stake in it.
HillaryClinton: "Saying hi at Whitney Brothers Inc., a furniture-maker in

Figure 5.7: The user interface of InsightsFeed [16] for analyzing Twitter data. (A) a list of tweets, (B) a sentiment chart, (C) user popularity chart, (D) a map from a 2D projection of tweets with important keywords highlighted in each region, and (E) feedback and controls over the progression and computations. The interface is progressively updated as more data is processed at a rate limited by the Twitter API used to download the data and by the analytical operations.

number of iterations for the *t*-SNE algorithm. In contrast to DimXplorer, where the progression is simply shown as a progress bar, this explicit widget permits the user to visit the history interactively.

It is worth emphasizing how InsightsFeed measures and communicates the stability of intermediate results. Unlike many PDA systems, which consider the amount of data processed as the main quality metric, the authors introduce a *quality metric* for the four views that increases with computation quality. For example, the metric for the tweet map is defined as the inverse of the projection error obtained from *t*-SNE. For the tweet list and the popularity visualization, the metric is defined as the number of new keywords/users added per chunk, respectively. These quality metrics are visualized over time as line charts in the control panel (Figure 5.8 top), and the user can move a vertical playhead on the line chart to return to a previous state and replay animation from there. The adoption of stability measures offers a clear example of how Implication 2 and Implication 3 can resonate with each other. For example,



Progress bar with absolute progress (current tweet, total tweets)
Quality (relative progress)
Playhead for history
15000
30000
Pause, stop (and show options)
Controls over computations and the progression
Controls for the progress and quality measures
Absolute Progress
Tweets Read
Relative Progress
Quality
Quality
TSNE Error
KMeans Error
#Clusters: 8
3
20
Chunk Size: 50
100
500
Waiting time (Sec): 1
1
10
Perplexity: 30
5
50
#Iterations: 200
100
500

Figure 5.8: Each view in InsightsFeed is augmented by user interface elements, such as a small widget that shows a stability metric and other widgets that allow the user to steer the computation.

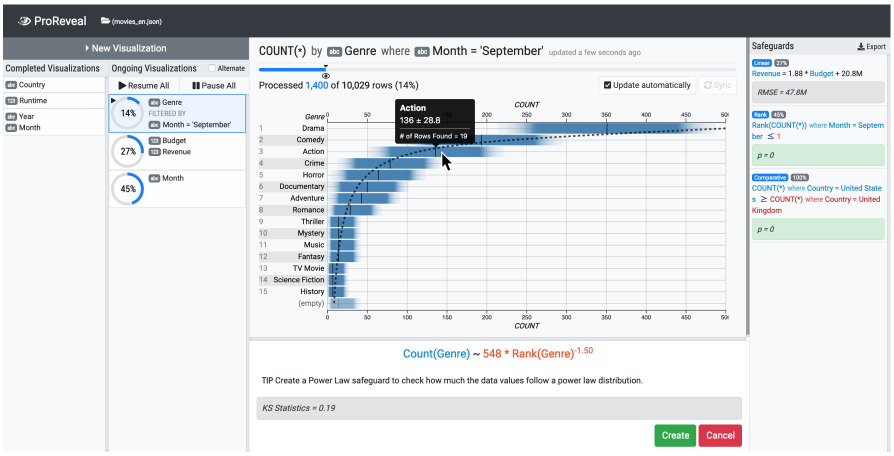
stability measures can be used to predict the amount of visual changes that will occur (Implication 2) and, at the same time, can serve as aggregated uncertainty indicators (Implication 3).

5.4.3 ProReveal

ProReveal [141] is a visualization system that implements a new exploration concept—Progressive Visual Analytics with safeguards—as a means of managing the uncertainty of intermediate knowledge during progressive data analysis.

In contrast to the two previous examples, in which the number of views on the screen was fixed, ProReveal allows the user to create new visualizations. This means that the number of running visualizations at a given time can change, and should thus share the limited amount of computation resources. ProReveal allows the user to set the priority between visualizations (R16) through drag-and-drop interaction with a priority list, which is a more sophisticated form of progression management than simple pause-and-play controls (R15, Implication 4). The user can also delay the reflection of partial results in the visualization to prevent abrupt changes (R3, Implication 2).

ProReveal provides a means of managing the uncertainty arising from progressive data exploration. During exploration, the user can interactively create *PVA-Guards*, a hypothetical or *speculative* representation of intermediate knowledge, on a progres-



Count(Genre) ~ 548 * Rank(Genre)$^{-1.50}$

TIP Create a Power Law safeguard to check how much the data values follow a power law distribution.

KS Statistics = 0.19

Figure 5.9: The ProReveal System [141] allows the user to add progressive visualizations on demand and manage their relative priority.

sive visualization. A PVA-Guard might represent the concept, for example, that “sales revenue is over 10000”. The PVA-Guards can be verified both online (in the middle of exploration) and offline (after exploration) without user intervention, providing a means of ensuring the correctness of the conclusion and of understanding the reason for intermediate knowledge becoming invalid.

ProReveal adopts two visualization techniques for visualizing uncertainty: gradient plots [59] and density maps with Value-Suppressing Uncertainty Palette (VSUP) [60] (R7, Implication 1). A gradient plot uses bars to visualize univariate values and the uncertainty associated with each value. In each bar, a value is shown as a black line at the center, and the confidence interval of the value is encoded using opacity; 95% two-tailed t confidence intervals are shown as fully opaque, while outside the intervals the opacity decays relative to the cumulative probability of an underlying t distribution (Figure 5.9). For density maps, ProReveal uses VSUP to encode value and its uncertainty to color simultaneously. In VSUP, the hue channel is mapped to value, and the lightness and saturation channels are mapped to uncertainty. When uncertainty is high, the hue range allocated for values narrows, deliberately making it harder for the readers to read the value and encouraging more cautious decision-making.

5.5 Challenges for Progressive Visual Analytics

In this section, we summarize the main challenges that are addressed in the literature, along with the considerations that have been raised in this chapter, which, if properly addressed, should make it possible to realize the scenario presented at the beginning of this chapter. **Research Agenda:** These challenges are relevant to researchers, practitioners, and stakeholders of PDA visualization.

- **Taxonomy of effective visual mappings for PVA**: More research is needed to explore the characteristics of existing visual mappings and their adaptability to progressive computation. Such effort should help to improve our understanding of what data characteristics, tasks, and degree of change during computation are supported by existing visual mappings. It should identify what aspects are critical to consider in adapting those visual mappings to PDA, while highlighting the combinations for which no existing visual mapping can be used. Ulmer *et al.* have provided an initial exploration of this space, identifying network and hierarchical visualizations as underexplored areas for progressive visualization. This may lead to specific extensions of the popular "Grammar of Graphics" (GoG) [332] and its implementations [266, 329] to better support PVA.
- **Research and novel visual mappings for PVA**: Based on the outcome of the previous challenge, research should focus on the creation of novel or "custom" visual mappings specifically created for supporting progressive visualization in certain domains. Those mappings could be derived from existing ones, perhaps even making significant changes to the original behavior in order to better support the visualization of partial results. Or, they could be completely new, derived from intrinsic characteristics of a progression that could be used as foundations to build on top of a novel visual mapping. Whatever kind is used, it should natively consider from the start interaction means and management of uncertainty.
- **Develop strategies for governing progression** From a visualization standpoint, PDA may have different ways of managing the progression that unfolds. Aspects that should be considered include: how to support the management of visual changes and visual update patterns; how to manage the user's visual attention and help her identify significant changes, or recover the current status of the progression, at any given time; how to manage needs concerning analysis time or the speed of production of new partial results. Finally, research should also focus on how to manage mid to long-lasting computation, where

the PDA process takes time to complete. Elements such as quick identification of changes relative to the last result seen and notification/context restore mechanism should be investigated.

- **Provide mechanisms for *attention management* in PVA** Research should focus on steering mechanisms for the progression that are able to represent both the "semantic" user desiderata and its adaptation to the sequence of partial results. Examples of aspects to consider range from managing the communication of progression metadata, giving priority to the smoothness of the unfolding, or highlighting "spikes," in both static (fixed) or dynamic (changing at run-time or by user selection) versions. Attention to a mechanism for integrating progression uncertainty into the visualization is also important, as it is possible to have good quality in a local area while the quality of the overall process remains low. Finally, support for speculative analysis and exploration of multiple concurrent progressive runs forming an ensemble are also aspects that need more visual support.
- **Expand the progression management on more than one dimension** As described in this chapter and revealed by Ulmer *et al.*, most of the existing approaches on progressive visualization apply the progression on a single dimension, which can be temporal, geospatial, etc. There may be cases where multiple dimensions require the application of progressive computation and visualization, such as dynamic networks, presenting both the management of changes over time and changes in the topology. Only one study [16] has explored this scenario to any degree, combining Twitter stream analysis with a changing *t*-SNE projection. Further research is needed to identify the most common combinations of dimensions on which to apply progression, how to make coexisting visual changes that come from these different dimensions, and how to effectively communicate those changes to the user in a way that is not confusing.

5.6 Summary

This chapter discusses the five ways of scaling visualization techniques: sampling, filtering, aggregation, multidimensional projection, and summarization. It explores the challenges of scalability and stability for visual rendering, and of controlling progress and steering for interactivity. The review of those aspects spans different phases of the design, from the choice of visual mapping to the interdependencies of the progression that unfolds with it, and finally, how to manage interaction with "progression-aware" interaction means. A description of three main contributions in the Progressive Visual Analytics field helps to understand what they mean in practice and what solutions might exist to address those concerns. Finally, we summarize recent challenges that researchers have encountered that might represent promising topics of research in the future. We hope the reader interested in designing a progressive visualization will benefit from this discussion of the implications to be considered during their design and from these example solutions.

6 Uncertainty and Quality

6.1 Introduction 92
6.2 Example Pipelines 95
6.3 Sources of *PDA Uncertainty* 96
6.3.1 Uncertainty from Data Processing 96
6.3.2 Uncertainty from Computation 97
6.3.3 Uncertainty from the Progression 98
6.4 Measures/Quantification of Uncertainty 99
6.4.1 Sampling Approximations 100
6.4.2 Aggregation Approximates 103
6.4.3 Computation-based Approximates 104
6.4.4 Progressive Measures 105
6.5 Visualization of *PDA Uncertainty* 105
6.6 Summary 107

Authors Anna Vilanova, Marco Angelini, Sriram Karthik Badam, and Jean-Daniel Fekete

6.1 Introduction

The ability to effectively model and communicate the quality and uncertainty of the current result is an essential component of Progressive Data Analysis. By definition, the intermediate results that a PDA system displays during a computation are not yet complete, and so are estimates of the final result [8]. Users must be able to evaluate the *PDA quality* or *PDA uncertainty* of the intermediate results to make well-informed decisions. The user has to decide if the information computed and presented so far is close enough to the final result to make a decision, or if more time is needed, as shown in Figure 6.1.

Definition 10 (PDA uncertainty)**.** PDA uncertainty *is defined in this chapter as the uncertainty about how likely the partial result is to deviate from the final result. Most processing pipelines generate results with uncertainties, e.g., due to noise in the data or parameter settings.* PDA uncertainty *focuses on the uncertainty that is introduced by the specific features of the progressive process to the intrinsic uncertainty of the results. It encompasses measures related to traditional uncertainty measures in statistics, such as* confidence intervals *(CIs) and Bayesian* posteriors, *that are used to measure the uncertainty of a single value generated from a specific function or endpoint (e.g., mean). It also encompasses uncertainty models used with more complex endpoints, data, and function types.*

Definition 11 (The PDA quality)**.** PDA quality *of a progressive result is defined in this chapter as the general degree at which any given partial result is a good approximation of the final result. A* PDA quality *measure is not a direct measure of the variability of the result, but is a more indirect measure intended to give a relative sense of how complete the current result is, e.g., the percentage of data samples processed.*

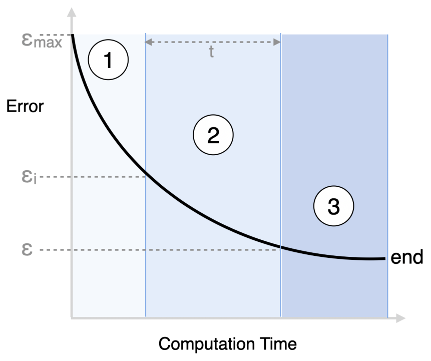
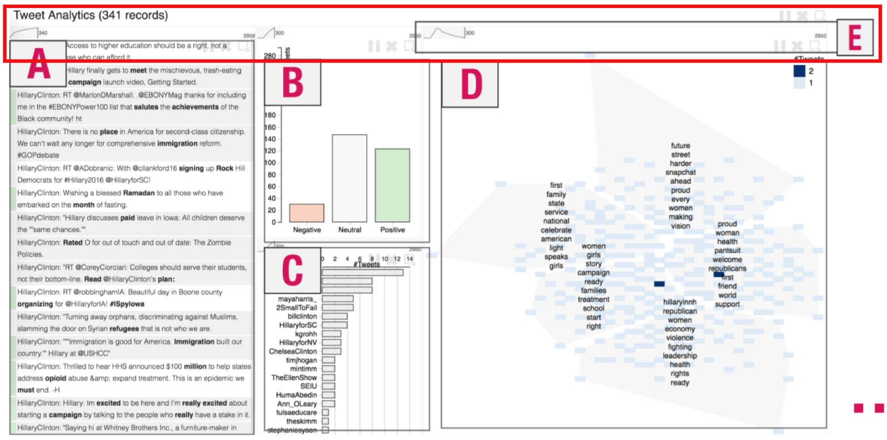


Figure 6.1: The three phases of PDA uncertainty. The vertical axis represents the error ε due to *PDA uncertainty*, which should be as small as possible at the end. The uncertainty ε should improve over time, from a (1) *chaotic* phase, in which the estimates are unreliable, a (2) *converging* phase, in which the estimates become more stable and slowly begin to show reliable patterns for decision-making, and a (3) *converged* phase, in which the estimates converge, signifying that decisions can be made accurately and the progression can end.

One of the main decisions that a user has to make is whether to act on an intermediate result or wait for the PDA to progress. Similar to Angelini *et al.* [8], we divide the process of a PDA analysis into three phases, as shown in Figure 6.1. In the first phase, the error ε due to *PDA uncertainty* is high, so the results are highly uncertain; there is a high risk when drawing any conclusion in this phase. In the second phase, intermediate results carry enough useful information to be interpreted. However, substantial *PDA uncertainty* remains and needs to be considered. During this phase, the user can make some predictions but cannot yet be sure of the results. In the third phase, the results are close to the final result, and further increases in quality will not influence the information carried by the results. The *PDA uncertainty* will not influence the analysis. In the final phase, the time spent on refinement can be considered wasted because there is no corresponding improvement in the results and uncertainty. An effective *PDA uncertainty* visualization both aids the user in determining the phase they are in and allows the user to assess the time needed to make well-informed decisions. *PDA quality* is used when we cannot model *PDA uncertainty*; it is a measure of the progress that indirectly relates to the quality of the results.

While visualization has long grappled with the implications of uncertainty in data [105, 140], *PDA uncertainty* and *quality* are intrinsic to PDA. By ignoring them, there is a real risk that users will make unfounded decisions based on premature results. For example, users might be tempted to stop progressivity at a point when the results confirm their own beliefs, but allowing the progression to run longer would have shown that the current result is an artifact. Conversely, users may take longer than needed to make good decisions if they cannot interpret the *PDA uncertainty* and *quality* correctly. Effective visualization of the *PDA uncertainty* and *quality* should aim at reducing or avoiding potential bias and errors as quickly as possible; the problem relates to the theory of sequential statistics [321] and *optimal stopping* [52].

The rest of this chapter offers an overview of *PDA uncertainty*. We begin with a motivating example to help illustrate how *PDA uncertainty* plays out in practice. Section 6.3 discusses the different sources of uncertainty in progressive data analysis, which are directly related to methods used to achieve the PDA strategy. Section 6.4 provides an overview of different methods that can be used to quantify *PDA uncertainty* and *quality* given the different axes that characterize them. Section 6.5 presents the different visual representations.



Access to higher education should be a right, not a
se who can afford it.

Hillary finally gets to **meet** the mischievous, trash-eating
campaign launch video, Getting Started.

HillaryClinton: RT @MarlonDMarshall: .@EBONYMag thanks for including me in the #EBONYPower100 list that **salutes** the **achievements** of the Black community! ht

HillaryClinton: There is no **place** in America for second-class citizenship. We can't wait any longer for comprehensive **immigration** reform. #GOPdebate

HillaryClinton: RT @ADobranic: With @cllankford16 **signing** up **Rock** Hill Democrats for #Hillary2016 @HillaryforSC!

HillaryClinton: Wishing a blessed **Ramadan** to all those who have embarked on the **month** of fasting.

HillaryClinton: "Hillary discusses **paid** leave in Iowa: All children deserve the ""same chances.""

HillaryClinton: **Rated** O for out of touch and out of date: The Zombie Policies.

HillaryClinton: "RT @CoreyCiorciari: Colleges should serve their students, not their bottom-line. **Read** @HillaryClinton's **plan:**

HillaryClinton: RT @robbinghamIA: Beautiful day in Boone county **organizing** for @HillaryforIA! **#ISpyIowa**

HillaryClinton: "Turning away orphans, discriminating against Muslims, slamming the door on Syrian **refugees** that is not who we are.

HillaryClinton: """Immigration is good for America. **Immigration** built our country."" Hillary at @USHCC"

HillaryClinton: Thrilled to hear HHS announced $100 **million** to help states address **opioid** abuse &amp; expand treatment. This is an epidemic we **must** end. -H

HillaryClinton: Hillary: Im **excited** to be here and I'm **really excited** about starting a **campaign** by talking to the people who **really** have a stake in it.

HillaryClinton: "Saying hi at Whitney Brothers Inc., a furniture-maker in

Figure 6.2: The user interface of InsightsFeed [16], same as Figure 5.7 with the *PDA uncertainty* and *quality* visualization aspects highlighted.

6.2 Example Pipelines

Let us begin by reviewing a sample system from the perspective of uncertainty. Section 1.3 introduces the InsightsFeed system [16]. In Figure 6.2, we see the system with four progressive visualizations, each dealing with different quality or uncertainty measures. They are unified visually using a simplified quality line chart over time displayed on top of the visualizations in the progress bar; the quality value in the line chart goes up when quality increases. However, the meaning of the quality measure is different for each visualization.

For example, in (A), the *PDA quality* is simply the percentage of tweets processed over the total, improving along with the progress bar. This particular screen shows that 340 of some 2500 tweets have been examined so far. In (B), the bar chart shows the distribution of the sentiments among the three categories *negative, neutral,* and *positive*. Its *PDA uncertainty* relates to the number of tweets in each category/bar, as later studied by Patil *et al.* [224]. (D) shows a *t*-SNE projection and its *PDA quality* can be seen as related to the objective function computed during the optimization process: the KL divergence [166] between the distances in high-dimension and low-dimension [311]. This is a mixture of *PDA quality* and the intrinsic quality of the final t-SNE result itself. In this case, it is not possible to separate the intrinsic quality from the *PDA quality* since the final value of the objective function is not known.

In this example, the quality visualization allows us to answer some simple questions: Is the process improving over time? Is it stabilized? If the quality is not stabilized, it is still in the first or second phase, and the current result might be inaccurate. If the quality of the results is not improving, it may be that it has reached the final stage and will not improve any further, or there is something wrong either in the data or the process (i.e., it is not shuffled, it has a strange distribution, it contains outliers or it does not converge).

Ideally, each visualization should provide a meaningful assessment of the *PDA uncertainty* or at least *PDA quality* computed from the data and the process, and in a form that allows users to make these decisions. For example, some dense areas of a t-SNE projection may become reliable sooner than sparse ones. *PDA uncertainty* and *quality* encompasses the choice of meaningful measures for each stage of a pipeline, their combination, and their visualization in a way that facilitates decision-making but does not hurt readability or users' attention.

6.3 Sources of *PDA Uncertainty*

Any data analysis system must wrestle with some forms of uncertainty: there are intrinsic uncertainties regardless of progressivity. Uncertainties and errors are intrinsic to the data, implying limitations in computation and visualization trustworthiness. Considerable attention has been devoted in recent years to the question of how to provide effective methods to communicate uncertainties in the visualization pipeline [105, 131, 140, 221, 287].

PDA uncertainty comes from the three main ways of making progressiveness possible: *data processing* (applies to *progressive-in-chunks*), *computation* (applies to *progressive-in-iterations*), and *the progression* itself. The goal of this section is to highlight these three sources and types of uncertainty. Once we identify the sources, we can quantify and visualize *PDA uncertainty* or *quality* to raise awareness among users and provide a complete picture for decision-making. Therefore, we will isolate the types of uncertainties.

6.3.1 Uncertainty from Data Processing

Fundamentally, the source of uncertainty in *progressive-in-chunks* PDA systems (introduced in Section 2.4) is that only part of the data is known at any point in the computation. Depending on the data sampling method, different assumptions can be made about the final results from the intermediate results.

Let's take the simple example of progressive bar charts, which are used in many progressive visualization articles (e.g., [16, 89, 141, 342]) and are studied by Patil *et al.* [224]. A bar chart is computed by aggregating values from categorical data, such as counting the number of tweets collected from a tweet feed, which are tagged as either *negative, neutral* or *positive*. When the aggregated values are computed progressively, the bars will not reflect all of the data. The intermediate result will have different uncertainties depending on whether the sampling method distributes the samples proportionally through the classes of the categorical attribute. In general, there are two aspects to consider with *PDA uncertainty*:

Sampling approximations lead to inconsistencies in the statistics presented to the user or sent downstream of a pipeline. The sampling strategy can define how the overall data statistics appear in data samples, as explained in Section 3.5.1. Random sampling can miss underrepresented entities in the data, while stratified sampling can be misleading because of high-level characteristics. In many PVA examples random sampling is used [16]; however, it is not uncommon to use the distribution statistics to define the progression [205], or to use data-specific sampling strategies [126].

Aggregation approximations are made when an aggregation function is applied to the sampled data. If this aggregation is not *distributive*, approximations are used. For example, the median operation is not distributive: the median of multiple chunks of values is not the same as the median of each chunk's median. Operations such as *median* are called *holistic*, since they require all of the data to be computed accurately [330]. Examples of useful holistic aggregation operators include the *median*, all the *quantiles* (except 0, and 100%, i.e., min and max), and the count of distinct values. Several of these holistic aggregations can be computed approximately using data sketches [57], but the approximation generates uncertainty in the results.

6.3.2 Uncertainty from Computation

Progressive-in-iterations methods (introduced in Section 2.4) create results from computations that approximate and incrementally approach the final computational outcomes. Some algorithms, such as optimization methods, have progression inherent to them. For example, *k*-means clustering is iterative until convergence is achieved based on a specific objective function that is optimized. Often, as in the case of *k*-means, convergence does not always mean that the best possible functional value is obtained—*k*-means algorithms are heuristics that often yield a local optimum, not a global one, so the functional end value is unknown. In this situation, it is difficult to evaluate the *PDA uncertainty* or *quality* independently of the intrinsic accuracy.

The stability of the functional value becomes a proxy for the *PDA quality* of the intermediate results as an indicator of convergence. Beyond the use of intermediate iterative optimization results, there are PDA methods that approximate computations. For example, computing the *k*-Nearest Neighbors approximately to accelerate the process (e.g., approximate *k*-Nearest Neighbors [142], progressive *t*-SNE [233], and progressive UMAP [160]). Here, more neighbors are computed progressively, creating more accurate results. In this way, the number of neighbors from k that are computed exactly increases through progression until all (or most) neighbors are exactly computed. Mühlbacher *et al.* [207] provide guidelines for creating such methods.

6.3.3 Uncertainty from the Progression

The progression itself, combined with the user and tasks, adds a layer of uncertainty. The evolving nature of the estimates and the continuous changes to the results can be challenging to follow. There are two types of uncertainty here:

Temporal uncertainty (see Figure 6.1): From a user's perspective, we can divide the progression of the results into three distinct phases of uncertainty, as explained in the introduction section. Overall, the user is interested in knowing which phase the progression is in and when the PDA results are good enough to make a decision, which is likely earlier than the time needed to reach the final result. Indicators presenting this information will be essential to the decision-making process of the user.

User/task uncertainties In addition, uncertainty may also arise from the user's task (see Section 7.3). The uncertainty can differ based on the nature of the task—hypothesis formation can occur earlier than the confirmation or communication of results. Ultimately, the user gets to decide when the result is good enough. As PDA designers, we should strive to identify all the types of uncertainty and provide as much information to the user as possible. Good PDA systems will help the user steer the process to the final results through the uncertainty phases as defined in Figure 6.1.

6.4 Measures/Quantification of Uncertainty

Now that we have defined the principal sources of *PDA uncertainty* in a progressive data analysis process, in this section, we provide an overview of the process of measuring or quantifying *PDA uncertainty* and *quality*. For example, we might want to present confidence bounds on results that are shown to users so that they can understand the accuracy of the approximation that appears at any given stage. Different parameters will influence the definition of *PDA uncertainty* or *quality* measures:

Absolute vs. relative measures These measures can be additionally considered with respect to their absolute values (i.e., a value per partial result with respect to the final iteration) or their relative value (i.e., difference between values for two generic partial results). For example, the objective function of the t-SNE algorithm is an absolute quality value that will converge to an optimum over the course of the optimization process. Conversely, the sum of the displacements of k-Nearest Neighbors centroids between two progressive iterations is relative.

Global vs. local measures A measure is global if a single value is computed for a whole partial result, and local when computed for part of the partial result or individual elements. For example, computing the stability of a k-Nearest Neighbors algorithm, as explained above, is a global measure, whereas computing the confidence interval of each bar in a progressive bar chart is a local form. In the latter case, a part is again considered for different stages of a pipeline (Data Processing or Visualization) or for different data intervals (e.g., a PDA process that computes a distribution of values can be stable in a fixed interval of partial results). Aggregation approximation strategies are a logical choice when you need to identify local versus global changes. For example, when aggregating a distribution by binning, the bin becomes the local unit. When aggregating a map across administrative boundaries (e.g., in a choropleth map), the administrative subdivision becomes the local unit.

Closed- vs. Open-world scenario A Closed-world scenario is one in which the total number of data points is known in advance, along with its characteristics (e.g., distribution). In a *progressive-in-chunks* system, for example, it means that the remaining number of data points to be processed is known. In a *progressive-in-iterations* system, it could mean the number of iterations that remain. In an Open-world scenario, on the other hand, the end target is not known, as, for example, with chunks read from a data stream of unknown size or from an iterative algorithm with an unknown end condition. The usefulness of a measure for uncertainty quantification will depend on whether we are in an Open- or Closed-world scenario.

Angelini *et al.*'s proposed a framework [8] for classifying the measures of intermediate results uses a three-aspect model: (1) *progression*, (2) *stability*, and (3) *uncertainty*. Here, each aspect allows the system to provide more precise estimates to the user regarding the potential *PDA uncertainty* or *quality* of the results available.

Progression: The requirement for this aspect is that it be possible to measure or estimate the amount of work remaining for the PDA to finish, i.e., a Closed-world scenario. A measure of progress can enable the user to estimate the likelihood that the actual results will not change much by the end of the process. *Progression* is the least effective measure of quality among the three, but it also serves other purposes (R9, R10, R11).

Stability: This refers to the degree of variation of the results produced by a PDA process. This measure can be used by the user as an indicator of how much the estimate of the final result is changing and to decide on that basis whether the current results can be considered sufficiently valid. We discuss stability with regards to convergence in Section 2.2. While unstable results cannot be reliable, stable results can be inaccurate. Therefore, it can be misleading to rely on stability alone.

Uncertainty: *PDA uncertainty* is defined as a measure of the degree by which the current partial result can still deviate from the final result (Definition 10). In Closed-world scenarios, when all the data has been processed, there is no more *PDA uncertainty*. In progressive scenarios (before completion), uncertainty estimation becomes a challenge. It cannot always be computed directly and conclusively.

PDA uncertainty quantification is a non-trivial problem since, in many situations, off-the-shelf solutions are not directly applicable to a progressive setting. Uncertainty quantification approaches can be classified according to the main types of progressivity described in Section 6.3. In the rest of this section, we present the different measures and quantification of *PDA uncertainty* and *quality* based on the sources of uncertainty.

6.4.1 Sampling Approximations

Sampling approximations are a widely used method in statistics for the study of uncertainty quantification (e.g., using confidence intervals and significance testing). The applicability of these methods is based on assumptions regarding the characteristics of the full data set from which we are sampling (i.e., Closed-world vs. Open-world).

If the overall sample size is known, a global *PDA quality* estimator is the ratio of the samples already processed and the total sample size of the full data set. This *PDA quality* estimator assumes that the more data that is processed, the more accurate the result will be.

Many systems use confidence intervals (CIs) as their indicator of uncertainty. For example, for the mean computation of a random variable, CIs are computed based on the central limit theorem. A confidence level of 95%, for example, means that the chances that the real mean value of the full population is inside the interval are above 95%. Therefore, there is a 5% chance that the real value is not inside the interval.

CIs for statistical parameter estimators, such as the mean, median, or total population, have been extensively studied in statistics. These CIs assume that the sampled data is independent and identically distributed (iid) and that there is a known overall data set sample size. In a PDA setting, these conditions are often violated and cannot be directly applied.

Uncertainty measures such as CIs can be updated progressively, but they introduce the problem of *repeated tests*. If humans want to make a decision based on progressively computed results and the associated uncertainty, they might carry out tests multiple times using the same samples. In our election example in Section 1.1, for example, a test to decide early who the winner is might be: does candidate A have significantly more votes than candidate B? While observing the progression of the count hour by hour, we would perform a test every hour. If we use a 95% confidence level for the test, there will be a 5% chance that we will make an error each time. Overall, the chances of an error will grow each time; if we conduct m tests, we will have a probability of error of 0.95^m, assuming the tests are performed with independent samples. But this is not the case; since they are cumulative. Therefore, the chance of making an error is slightly different, but the principle is the same: we need to correct the test each time to keep the confidence level constant. The statistics related to the problem of compensating for multiple tests have been studied in the domain of sequential analysis [321].

Sequential analysis methods are used in applications where, for example, the cost of data collection is high, or there are ethical reasons for not collecting unnecessary samples. Instead of having a fixed sample size, the tests are done sequentially and stopped when there are enough samples. Popular methods to address the *optimal stopping* [52] problem and making the right decision include SPRT [321], Haybittle–Peto, and O'Brien–Fleming boundary tests [218, 236]. However, these are complex to use in an exploratory setting.

Distribution bias is another important issue in statistical analysis that is related to the difficulty of making decisions when the distribution of the samples is not known or is not well-behaved. In that case, guessing the distribution is also difficult, and there is no magic method for creating a black-box sequential significance test [224]. Several tests can be used heuristically, but aside from a few special cases, e.g., normal or well-behaved distributions, they will all have an unpredictable error one way or the other. Therefore, it falls to the human user to arrive at a final answer. Returning to the election data, we are aware of the non-stationary sequence for counting the ballots. Cities have different distributions than rural areas, foreign voters are different from domestic voters, etc. We have to allow for all these factors whenever we attempt to predict the outcome of an election before the counting of ballots is finished. But sometimes, even with all of our experience, our attempts fail. And sometimes the statistics methods themselves fail, depending on the confidence level requested.

For a histogram estimator, simple known distributions after a certain number of samples of the underlying probability distribution can be fitted and guessed. The accuracy can also be measured by a confidence interval per bin, although by doing so, there is a dependency that is ignored. Patil *et al.* [224] have studied sequential analysis for a few tasks on bar charts. They report that humans are quite good at finding a good trade-off between wait time and accuracy in making decisions. They also show that statistical tests can be used, sometimes with good results on their experimental data, but without any formal guarantees. They also attempted to infer data distributions empirically, but concluded that this is generally difficult and unreliable.

When the distribution is unknown or the estimator or data is complex, non-parametric Monte Carlo simulations or bootstrapping methods [76] can be used to evaluate uncertainties. Monte Carlo simulation and bootstrapping methods are more flexible and can be used for complex functions as parameter estimators (i.e., processing pipelines) that have no analytical solutions. They are based on simulation techniques that use resampling strategies to obtain multiple possible output samples that describe the variability of the output. These multiple output samples often need to be modeled or summarized to be communicated and to adequately represent the output uncertainties [86, 285]. However, they are computationally costly [340] and challenging to model since assumptions, e.g., on fitting parametric distributions, do not hold.

In summary, while an extensive amount of work exists in statistics on the definition of errors and confidence intervals, for complex real-world PDA problems, the necessary assumptions for these methods will be violated and out-of-the-box methods will not be adequate.

Research Agenda: The implications of using traditional statistics are unclear when certain assumptions, such as the iid, are violated, as is often the case with PDA problems. The use of sequential analysis methods in PDA solutions appears to be an under-explored area.

Research Agenda: High-dimensional complex data analysis and complex estimators remain an open problem with regard to *PDA uncertainty*. There is no unique way to indicate the uncertainty of complex data beyond well-defined high-dimensional numerical spaces.

6.4.2 Aggregation Approximates

Aggregation approximates are used in progressive refinement approaches, in which there is typically a hierarchy defined that aggregates data at different levels of detail in a pre-processing step. In these approximations, we are often in a Closed-world scenario. For example, a volumetric multiresolution grid (e.g.,, octree) is built to accelerate simulations or volume rendering, or a data hierarchy is built to explore high-dimensional data (e.g., HSNE [232]) at different levels of detail. These methods can use traditional statistics and summaries that indicate the level of approximation at each hierarchy node. The estimation of uncertainty can either be local or an aggregation to indicate global. A *PDA quality* indicator can also be the level in the hierarchy that is being analyzed.

For example, processing GPS data at country-, city- or neighborhood-level aggregates creates approximations that follow progressive refinement. A simple measure of uncertainty is the level of aggregation being processed at the most detailed level. It is often possible to compute the information that is lost on the data aggregation based on the variation of the elements that form it. This measure shows, either locally or globally, the uncertainty of the distribution of error in the input data. Processing the GPS locations is done with grid-based approximation (approximating them to the nearest grid location) or creating routes inside them using simplification algorithms (e.g., Douglas–Peucker algorithm [73]).

However, the analytical propagation of the uncertainty in the input to the output result can only be done in special situations and not in general cases. For that reason, the direct quantification of output uncertainty is often challenging, and Monte Carlo or bootstrap methods combined with heuristics will be used.

6.4.3 Computation-based Approximates

Computation-based approximates are a common way to accelerate processes in which the computational complexity itself is high. In these methods, the computations are approximated to accelerate the process or to ensure that it is possible to obtain any result at all. There are multiple examples of these approaches. In visualization, for example, progressive dimensionality reduction implementations of t-SNE [142, 233, 311] or clustering methods such as k-means are commonly used. Randomized computations, such as Monte Carlo approximation [343] for numerical approaches, are a common example of approximation. These methods usually have clear theoretical error bounds that can be computed analytically. Heuristic approaches are also used, for example, by using the locality of a problem. In that case, the bounds are harder to define. Error tolerance can be allowed, for example, approximating instead of computing an exact k-Nearest Neighbors, which requires the computation of the distance between all pairs of dataset points. Using the locality of the problem, the neighborhood can be approximated with a predefined uncertainty level (i.e., the number of k-neighbors computed accurately). This level can be progressively refined [233]. The measure of uncertainty in the output (e.g., distance computation) cannot be computed. However, a local indicator of *PDA quality* is the number of neighbors from k that have been exactly computed per data point. In this case, we have a clear global endpoint in which all k nearest neighbors per data point are exactly computed.

Optimization algorithms, e.g., based on gradient descent [173] lean easily towards progressivity, given their iterative nature. The evolution of the optimization or cost function provides an indication of the *PDA quality* of the result [16] and, therefore, it is not possible to discern the *PDA quality* from the intrinsic quality of the final result. However, it remains unclear at which exact value the minimum or maximum will be achieved, and it is not adequate as a measure of uncertainty. The evaluation of the value would be expected to be similar to the one presented in Figure 6.1. Unfortunately, the absolute values of the error ε are unknown in advance. It is not, however, an uncertainty measure. Stability would relate to when the current objective function values and the ones in the past iterations are not changing, which is a common measure of convergence. For example, the well-known k-means clustering algorithm is NP-complete to find the optimal solution, so its implementations are heuristics that generate errors and uncertainty on the cluster members and cluster centroid locations. Progressive implementations, like most implementations, use the stability of the cluster centroids as quality, but they cannot show the bounds or represent the uncertainty in the form of, for example, confidence intervals.

6.4.4 Progressive Measures

In principle, in a PDA process, the *PDA uncertainty* or *quality* indicators are computed synchronously with every partial result. However, if the computation of a measure is expensive, it could also be computed in a progressive fashion and thus appear asynchronously. For example, using bootstrapping to compute CIs is iterative and can be done in a progressive way for complex computations.

Research Agenda: The *PDA uncertainty* or *quality* would appear with a delay or might force the visualization of the result to be delayed. More research is needed to better understand how users can interpret *PDA uncertainty* or *quality* measures when they appear to be delayed.

6.5 Visualization of *PDA Uncertainty*

After finding the sources of uncertainty and ways to quantify them, it is important to visualize this information to the user in an unbiased and adequate way. In this section, we present visual encoding strategies for *PDA uncertainty* and *quality* as a complement to Chapter 5.

General guidelines exist for visualizing uncertainty in the research literature [131, 140, 221]. Standard visual variables offer a good starting point, but they are often already used to convey the data itself. Position, length, and color are popular choices in many progressive systems for conveying uncertainty. Examples include, (1) utilizing heatmaps to convey uncertainty in position [16, 293], (2) error bars to convey uncertainties in value and distribution of variables [224, 342], (3) transparency and color saturation to convey uncertainty in bi-variate distributions [141, 205, 293], and (4) using multiple visualizations to couple data insights and uncertainty separately [16].

Visualizing uncertainty goes hand-in-hand with providing a sense of *PDA uncertainty* or *quality* in the progression, as described in previous sections. Stability in PDA corresponds to the amount of change in the visualization or, rather, the outcomes from the underlying computation or data sampling. When progressive visualizations are updated, the change between frames is visible as an animation. Therefore, animation is the most straightforward way to indicate stability. On the other hand, quality can be quantifiable from the underlying computation in terms of the error and accuracy estimates provided by the algorithms. For instance, when using k-means, the centroid shifts can be visualized as an indicator of the stability of the computation [83].

Differences in PDA vs. general uncertainty The main novel issues related to *PDA uncertainty* are the need to monitor the progression, time-evolving stability, and quality or uncertainty related to PDA. Avoiding bias in the users by showing progression results is also a major subject of interest.

For the progression, most PVA systems use some sort of progress bar in the interface. InsightsFeed introduces specific UI elements to show the progression and, on top of it, the stability as a line chart of quality (see Figure 5.8). The TensorFlow Playground [298] also visualizes the loss over time while running a neural network learning session. Discrete/unit visualizations are preferable for conveying uncertain outcomes in related work [130]. This is because unit visualizations can provide an open representation of the units within the data. Hence, in contrast to continuous/aggregate representations such as line charts and bar charts, discrete representations have an innate sense of uncertainty. Dot plots [84] and density maps are examples of discrete visualizations. These plots make generous use of visual marks to show individual data points (or a few data points), making it possible to keep track of sampling uncertainties.

Unfortunately, in many cases, progressive visualizations do not have the luxury of representing individual data points due to the sheer size of the datasets. Therefore, progressive visualizations face challenges in providing a complete picture of the uncertainty in the progression, outcomes, and user understanding.

Research Agenda: More research is needed on the question of a systematic approach to conveying progression, quality, and stability, along with interactive controls over the progression to assist users.

Uncertainty/quality changes over time Similar to the progressive outcome itself, it is important to visualize the changes in uncertainty and to provide the user with controls over the progressive outcome, as highlighted by the requirements of PDA systems listed in Table 2.1. Animation is commonly used to show the progression. However, animation has its limitations, such as a tendency to direct too much of the users' attention to small changes, thus overloading short-term memory. In some progressive systems, the presence of multiple progressive visualizations can be challenging for the user [205]. In such a situation, tracking absolute and relative progress for each progressive visualization is important. The uncertainty measures have been conveyed with line charts in the relevant literature [16,83]. Basic controls to pause, replay, stop, and refresh the progressive visualizations, and advanced controls over the parameters of the underlying visualizations, offer ways to understand and deal with the *PDA uncertainty* outcomes from the progression using animation [83].

Showing *PDA uncertainty* and *quality* is essential, but translating them into decisions is another important step in the process. A PDA system can help in this regard. It can either provide advice on how to read and make use of uncertainty, as explained in the training from Patil *et al.* [224], or by adding tools to support decision-making, as explained in Ferreira *et al.* [85] and in the ProReveal system [141]. While these tools can mitigate possible cognitive biases (see Section 7.5), they should not be blindly trusted, as the existing tests are sensitive to difficult data distributions, outliers, and biased sampling. **Research Agenda:** More research is needed on the question of how to better guide users in interpreting uncertainty/quality over time for decision-making.

6.6 Summary

In this chapter, we presented multiple classifications and ways of presenting the uncertainty that is an intrinsic part of a PDA system, *PDA uncertainty* and *quality*. We introduced three phases of uncertainty (chaotic, converging, and converged), while defining the characteristics and expectations of uncertainty during the PDA process. The multiple sources of uncertainty are presented: sampling for *progressive-in-chunks*, convergence for *progressive-in-iterations*, and many others arising from the progression itself. We classified the different characteristics of the uncertainty measures and quantification based on the assumptions and knowledge of the specific PDA situation described in the literature. Most measures that are well-established, such as using CIs, are based on assumptions that often are not applicable for testing in a PDA setting, and alternatives like the ones proposed by sequential analysis should be considered. Finally, the challenges and existing strategies for the visualization of *PDA uncertainty* and *quality* were presented. The characteristics of progressivity add another layer of complexity and additional constraints to traditional uncertainty visualization and thus call for more research.

7 Human Aspects

7.1 Time Engineering for Human Factors 110
7.2 User Roles 111
7.2.1 Role 1: The Progressive Observer 111
7.2.2 Role 2: The Progressive Searcher 112
7.2.3 Role 3: The Progressive Explorer 113
7.3 User Tasks 114
7.3.1 The Observer's Analytic Tasks 115
7.3.2 The Searcher's Analytic Tasks 116
7.3.3 The Explorer's Analytic Tasks 116
7.4 User Focus 117
7.4.1 F1: Focus on the Data Space 118
7.4.2 F2: Focus on the Algorithm Space 119
7.5 User Complications and Biases 120
7.6 Application Example: Crisis Management 124
7.6.1 The Background 124
7.6.2 An Illustrating Scenario 126
7.6.3 The Design Guidance 128
7.7 Summary 130

Author Hans-Jörg Schulz, Michaël Aupetit, Danyel Fisher

From a user-interaction perspective, what makes Progressive Data Analytics different from classic visual analytics is that it introduces a new temporal rhythm to data analysis loops. Following Dimara and Perin [69], we understand interaction for visualization to be "the interplay between a person and a data interface involving a data-related intent, at least one action from the person and an interface reaction that is perceived as such." Time perception is crucial in that interaction dialogue, and time reaction expectations can differ depending on the context [42, 180].

We turn in this chapter to five aspects of human interaction in visual analytics:

Time Engineering: We discuss temporal implications for user interfaces. In contrast to eager systems, where we simply try to design interfaces that are as fast as possible, progressive systems offer us design opportunities for maintaining interactivity.

User Roles: We think of two concurrent processes: a computer system carrying out computations, and a human observing them. Depending on how often and how much these two concurrent processes inform each other, PDA users can have different levels of involvement in the progression—from merely monitoring the stream of results like viewing an animation to actively steering the progression along a computational path of choice.

User Tasks: Managing how these two processes interleave requires users to perform various additional tasks that are not needed in non-progressive analysis setups. For example, users need to gauge the uncertainty of an intermediate in-progress result (see also Chapter 6). They also may want to refine the search space of a progressive analysis process after making an observation in an intermediate output.

User Focus: Different motivations can be behind the use of PDA. Some users may employ the progression to come to a speedy conclusion about a dataset under investigation, while others may use the progression to interactively explore differently parameterized computations by observing and comparing their progress while they run.

User Complications and Biases: Given the greater influence of the user over the computational process, users' subjectivity and biases have a higher potential of affecting the analysis (see also Chapter 9). This includes, for example, the possibility of cherry-picking results by terminating the computation just at the right moment while it is showing a desired output.

Our goal is to better understand how and when a PDA system can be useful (or harmful) and how different PDA techniques can be used to address different user needs. This characterization is adapted in part from Micallef *et al.* [197]. We believe that articulating this set of personas and tasks can help with both the design of, and research on, progressive systems. The aspects identified and discussed here might be useful for:

- application designers, to identify potential user needs; to consider users taking different or supplementary roles to their current one to benefit from the full scope of PDA techniques; and to explore alternative solutions afforded by PDA to inform their final proposal;
- software engineers, to take inspiration from the literature linked to the PDA usages specific to each role, to derive the best technical solution to support progressive tasks for their respective users;

- domain experts and end-users during a collaborative design process [337] with PDA designers to identify and discuss the potential pitfalls and determine the actual risks of such biases, their impact, and ways to mitigate them;
- PDA system designers to provide frameworks for PDA designers to ensure that PDA systems are evaluated with respect to the different types of usage by covering relevant user roles and tasks; and to propose to software engineers PDA workflows providing adequate methods and tailored visual details for typical progressive usage scenarios specific to an application domain.

We conclude this chapter by bringing together these themes in a single usage scenario showing how these concepts can guide a designer in charge of improving a crisis management system.

7.1 Time Engineering for Human Factors

We begin with a broad observation about building interactive tools. One major goal of PDA is to ensure users can maintain an interactive experience, even across large datasets and complex computations. To do so, it is valuable to understand what "interactive" means.

A user will perceive an interface as "interactive" if it responds in under a second; an interface that doesn't show updates or changes within ten seconds will likely seem frozen [42, 180]. Different interface designs can build expectations of responsiveness: a visual progress indicator can help show updates, but will seem very slow if it does not make visible progress; while a numerical indicator can help maintain an appearance of change. In contrast, a user might expect a slider to respond interactively, while they might be prepared to wait for a response to a button press [281].

Interface design also drives users' expectations with the system. An interface that responds quickly invites exploration, but also quick judgment; it also risks having too many things change too quickly. Unless the visualizations are stabilized (see Section 5.2.1), the rapid changes may disorient the user. Conversely, an interface with a higher latency encourages a user to be thoughtful about firing off a new query, but therefore discourages freewheeling exploration. With high enough latency, a user is likely to switch tasks, multitasking while they wait for a slow job to finish.

Progressive Data Analysis aims at handling these time constraints by presenting meaningful partial results to the user early and frequently enough before full completion—and the user can, in turn, respond concurrently, steering or tweaking the computation as it runs. This makes for a tighter, more interactive analytic loop.

7.2 User Roles

At the most basic level, while a user is in the process of carrying out data exploration with a Progressive Data Analysis system, they will issue a series of queries, then wait for the system to produce results. There are a number of ways that a PDA system can allow for increased user involvement, from being able to indicate the internal state of a computational process (e.g., stalled vs. running) to actively steering the computation toward subspaces of interest while it is running (e.g., data subspaces, parameter subspaces) [207]. Here, we draw a distinction between three user roles characterized by their increasing involvement in the PDA interaction loop. These roles range from the more passive *Observer*, to the goal-oriented *Searcher*, all the way to the free-reigning *Explorer*.

7.2.1 Role 1: The Progressive Observer

The Observer's main interest is in monitoring a progressive process, often without having any knowledge of that process or of PDA as an underlying concept. Thus, the Observer mainly wants to stay informed about the state of the computation for confirming its aliveness (R9) [207], for estimating how long it will take until it is finished (R10), and for establishing a sense of provenance of the result (R8) by seeing it unfold. An Observer might, for example, watch a progressive treemap [252] being refined with more data over time. The ability to watch the evolution of the treemap from a single top-level rectangle that is then further and further subdivided instills an understanding of the generation process, gives an indication of its progress, and helps the user learn about the hierarchy as internal nodes are drawn before their children. Tracking this process lets them understand the data they are visualizing, the algorithm being used, and gives them a continuing sense of progress.

An Observer's expectation of the underlying PDA process is a steadily improving output that arrives at a predictable end. This is familiar from applications that show gradually-refining images, or from loading progress bars. To support an Observer in using PDA without an extra learning curve for understanding its progressivity, a PDA system can be tailored to meet the Observer's expectations by the following affordances:

A. Reduce visual complexity, fluctuations, jumps, flickering, or any other indication of the underlying process not smoothly and predictably producing progressive outputs—in particular if these are induced by external influences (garbage collection, memory swapping, etc.); they also relate to design requirements R3 and

R4 (see Chapter 2).

B. Use commonplace metaphors and familiar abstractions that are already part of the interface—e.g., that of a progressively loading “online map application”, as we often see in large graph visualization [213].

C. Maintain the users’ mental map, e.g., by introducing anchors [16] or by slowing down fast computations [314].

7.2.2 Role 2: The Progressive Searcher

The Searcher’s main interest is using PDA to quickly find an answer to a concrete question or a solution to a particular problem in a large amount of data and information. To that end, the Searcher usually comes with a specific, run-time-intensive query that shall be processed in a resource-conscious way. For example, this resource may be time, when basing an urgent decision on the query result. It may also be usage of computing infrastructure and the incurred costs for running it (e.g., electricity, network traffic, cloud billing), which shall be cut down to the necessary minimum. Hence, the Searcher is after a trustworthy intermediate result that is just good enough for the purpose for which it is needed, permitting them to cancel the running query, saving the system a costly exhaustive search. The query is not confined to the data space, but can also entail a search among algorithms and parameters to quickly identify promising computational models. Thus, Searchers use PDA mainly for terminating processes mid-run in an informed manner—either to dismiss erroneous queries or to terminate the progression ahead of time once a good enough result is computed. Both of these can be leveraged to guide the search process by subsequently eliminating bad search paths in order to reduce the search space, effectively speeding up the overall search spanning multiple queries. A Searcher’s expectation of the underlying PDA is that of a step-wise process that produces constantly updated results, which can be terminated at any time to readjust the computation or use its intermediate result instead of a final result for subsequent analytic operations. To facilitate the needs of a Searcher, a PDA system can employ the following affordances:

A. Reflect—maybe even emphasize—any fluctuations, jumps, and other indications of the underlying process to make the user aware of possible issues with the query or model (also relates to R5);

B. Indicate uncertainty in a domain-specific and visual manner that allows judging of the reliability of the intermediate results [8, 10] (R7, see also Chapter 6);

C. Provide the option to subdivide large queries or model runs so that each sequential step in them can be run and checked individually for success.

D. Provide the option to parallelize queries or model runs (see Section 3.4), yielding an ensemble of progressively refining outputs that can be compared for their agreement [238].

7.2.3 Role 3: The Progressive Explorer

The Explorer's main interest in PDA is to get to know their dataset rapidly, and to explore questions that they may not have formulated clearly yet. It is often not a single result in which the Explorer is interested, but rather in gaining a comprehensive understanding of data and process. They may wish to get to know distributions of their data, or understand how different dimensions relate to each other. Hence, Explorers use the intermediate results of PDA to orient themselves and to decide in which direction to steer their "tour" through the data or parameter space, swiftly adjusting queries, views, runtime constraints, or all three simultaneously [66] based on what they observe in the intermediate outputs.

Their expectation of PDA is that of an open, malleable process that can be redirected, re-parametrized, re-prioritized, refocused, and interactively perturbed. Explorers are aware of the progressive nature of the underlying process, as well as the fluctuations and discontinuities their interactive adjustments introduce into that process. In fact, it is often these repercussions of their actions that they are interested in, as these may reflect underlying data properties or functional dependencies. To support such an involved use, PDA systems can offer affordances like:

A. Expose details and parameters of the PDA system to the user and make them adjustable on the fly;

B. Incorporate summaries, statistics, and overviews to provide a global context for evaluating current assumptions and insights, if possible automatically [141];

C. Allow possible changes in a What-If manner that branches off alternatives and compares them later [1].

Users might switch roles during a PDA session—for example, a Searcher might start off wanting to run a quick progressive query, before discovering that the running query does not report back sensible intermediate results. That might prompt them to shift to an Explorer role to test different parameter settings on the fly, to "help the query along". **Research Agenda:** How to recognize the need for such a switch and how to facilitate these switches is an open research question. In essence, a PDA system should only expose as much complexity and functionality to the user as is currently needed, but how to do this best and as seamlessly as possible is not a settled question. Does the Observer automatically become a Searcher the moment they start interacting with the progressive outputs? Do switches between user roles only go in one direction (e.g., from less to more involved in the progression) or are there also scenarios in which it would make sense to “down-grade” an Explorer to an Observer if they have not interacted in a while?

From the perspective of designing PDA systems, increased user involvement can incur greater computational overhead. This overhead stems from providing the necessary interactive flexibility to change and re-parameterize the progression on the fly, as well as from ensuring that the progression can always be interrupted. Many of these additional complexities of progressive data handling and computations are discussed in Chapter 3 and Chapter 4. When designing a PDA system, it thus makes sense to strike a sensible balance between desired user involvement and acceptable overhead.

7.3 User Tasks

In general, a PDA approach can support the same user tasks as described in the visual analytics (VA) literature (e.g., [35] or [263]). The progressive nature of the process might influence and change a user’s workflow. These changes might range from expanding the task scope to understanding the semantics of tasks differently.

A certain coupling arises in this approach between an understanding of the data (e.g., compare results of analyses, explore the data overview) and an understanding of the underlying progressive process (e.g., comparison between partial results, evaluation of uncertainty of the progression, exploration of a succession of partial results). This coupling motivates the reformulation of these tasks in a way that respects the progressive nature of the process but also becomes a primary concern of the analysis.

For example, tasks that usually have clear answers are transformed into probabilistic queries: “is column A taller than column B” becomes “how likely is it that column A is taller than column B”? [85]. This, in turn, introduces the additional task of evaluating uncertainty: “Is it certain enough that A is taller than B to proceed with our analysis?”

In the following, we give a set of characteristic analytic tasks for the different PDA user roles and their respective affordances (see Section 7.2). These advanced tasks come in addition to the basic ones required to deal with the progressive process—e.g., running, pausing, terminating the computation, or increasing, decreasing the step size/chunk size, judging the overall quality of intermediate results, or identifying what has changed between visual updates.

Research Agenda: None of the tasks below directly explore the meaning of progression for the classic visual analysis tasks. As such, it is still an open question how to adapt Shneiderman's Visual Information-Seeking Mantra [282] "Overview first, zoom & filter, details on demand." for progressive visualization. In PVA, we do not have a complete overview of the data to start from, let alone any details that may come only at the very end of a progression. How do we best support visual information-seeking on incomplete, intermediate results?

7.3.1 The Observer's Analytic Tasks

Although the Observer's role in PDA is typically passive, they will still aim to achieve analysis tasks—be it to simply monitor the progression to understand the build-up of a large information space [253], or to track individual patterns such as forming clusters, which are also important tasks in streaming data scenarios [251]. Specifically, we note that an Observer would use PDA to perform the following actions:

T1. Ascertain suitable quantity of processed data and stability of the computation results (RO1-A) for subsequent analysis steps. In situations where minimally viable results need to be available quickly, it might be helpful to monitor the sequence of progressive results to ensure these scenario-dependent constraints are met [8, 10].

T2. View large information spaces progressively (RO1-B) as complex datasets, computation results, or visualizations are built up incrementally. This kind of monitoring of the build-up of complex results allows for forming and keeping a train of thought alongside the running progression, instead of being confronted with a possibly cluttered end result in one go. This task is supported, for example, by Progressive Parallel Coordinates [254] that build up step by step into the full and usually cluttered view showing all the data.

T3. Understand an algorithm and its inner workings (RO1-C) by observing its output and internal state as it runs, for instance visualizing a sorting algorithm [187]. This is important when ensuring the provenance of computational results, which ultimately builds trust in the otherwise hidden computational part of the visual analytics process.

7.3.2 The Searcher's Analytic Tasks

Searchers may perform all of the Observers' tasks to monitor the progress of their search. In addition, they actively influence the progression through their queries and parameter adjustments, which requires carrying out additional tasks. As the Searchers' analyses involve a specific question to be answered, their tasks reflect the notion of a sequence of coupled actions and reactions in the sense of a dialogue between analyst and progressive process, as put forth by Turkay *et al.* [304]. We list four common tasks for the Searcher:

T4. Analyze an approximate or partial result of a costly query (RO2-B&D) as one may not be able to wait for the arrival of the final result [89]. This analysis should be done with the explicit knowledge that the analyzed result is uncertain and the actual final result may differ.

T5. Refine search space based on the intermediate results provided by the progression (RO2-D), so as to make the progressive results more meaningful by targeting a given parameter or data subspace of interest [66, 123]. This task aims to specify a search objective in a number of quick, iterative steps based on early outcomes of the progression in order to yield more pertinent partial results earlier.

T6. Compare different executions of the computation (RO2-A&C) to analyze the effects of different parameter settings or input data on a computational method, so as to find the search term or the parameterization of the search that yields the best results [181].

T7. Stop a costly progressive computation not showing promising results (RO2-B) to stop a computation early when trying to perform a costly operation when there is no hope of the operation yielding satisfying results, saving time and energy.

7.3.3 The Explorer's Analytic Tasks

Explorers must be able to perform all the tasks of the Observers and the Searchers. But since they also need to actively steer the progression and interact heavily with the underlying progressive process [210], they also need support in the following tasks:

T8. Gain an overview of an unfamiliar information space (RO3-B), as often Explorers are dealing with datasets or parameter spaces for the first time. Hence, they need to be able to get a quick first glance at the big picture to orient themselves before steering the analysis in a particular direction [275]. This may involve visualizations of distributions of different dimensions, or the use of a "breadth-first" style progression of samples of the full information space typical of cluster visualization of search results [45, 46].

T9. Identify possibilities for furthering the computation by integrating the user's tacit knowledge or preferences (RO3-A) at different stages of the ongoing computation. This might involve, for example, influencing an otherwise unsupervised algorithm by interactively adding tacit knowledge [269] in clustering or considering *Must-Link* and *Cannot-link* constraints [320], or it might involve interactively sampling a high-dimensional energy function through its low-dimensional projection [72].

T10. Investigate alternative scenarios (RO3-C) by branching off the analysis into What-if scenarios, pursuing them in parallel, only to then decide for the better one in the end or to merge their results [325]. Because of the progression, this can actually be done mid-computation; as of T7, less fruitful analyses can be stopped early.

7.4 User Focus

Due to long execution times often inherent in eager visual analytics, it is common to separate the visual analytics process into two stages: a planning phase, in which the analytics process is planned in the *algorithm space* by choosing and configuring the computational methods, models, and parameters; and an execution phase, in which the configured analysis workflow is carried out in the *data space* [338]. This separation is so ingrained in current practices that it is only explicitly mentioned in a few ambitious cases aiming to combine both, e.g., [165, 268]. This distinction is less relevant in PDA, and users can and should switch seamlessly between investigating the data and adjusting the algorithm while the analysis runs. Yet because PDA increasingly blurs the lines between these two analytic foci, considering them explicitly for the design and evaluation of PDA systems becomes even more important. Mülbacher *et al.* has proposed the notion of *entity of interest* [207], which distinguishes between the execution of a computation and that computation's results (intermediate or final). In that spirit, the following classification distinguishes between the data being the focus of the progressive visual analysis, or the algorithms and the process of running them.

7.4.1 F1: Focus on the Data Space

When focusing on the data, analysts want to make use of PDA for analyzing their data without necessarily having to understand the internal workings of the progressive algorithms. This means that decisions about chunk size, feedback about the stability/certainty of the intermediate result, potentially setting "progressive guards" etc. that require understanding the underlying progressive processes, are not within the scope of the Observer.

In the simplest case, the Observer (RO1) can **Monitor the Data (U1)** for staying informed about an ongoing background process in the data space—whether it be a computation or a data transmission. This information includes, for example, the progression's aliveness and progress. A prime example of this kind of use is *visual sedimentation* [132], which renders otherwise invisible data updates as a trickling stream of particles settling down into a preliminary result.

This communicates the aliveness of the background process, the speed at which the data comes in, as well as the fact that the current visualization is not yet final. Building up a visualization in such an animated manner can also be used to mitigate the problem of cluttered visualizations. For instance, a thoughtfully prioritized progressive loading of different predetermined aspects of a data set (e.g., outliers or subtle trends) can ensure that such data aspects are shown during intermediate steps before they disappear in the clutter of the full visualization [253].

The Searcher (RO2) can **Query the Data Progressively (U2)**, to query a very large data set when they are not able to wait for the timely and costly data retrieval computation to return the accurate results. Because PDA makes intermediate results usable and explorable as if they were already complete, the searcher can start inspecting query results as they appear. The way in which the Searcher zeros in on some parts of the data (e.g., through zooming, panning, or selection) while disregarding others data subspaces can then be used to refine the query to better reflect the user interest expressed in these actions [66, 125].

The Explorer (RO3) can **Explore the Data Progressively (U3)**, they make use of PDA to provide an overview even of large information spaces that refines over time and enables early interaction, such as zooming into regions that seem particularly promising. This, in turn, can direct the progressive computation to the parts of the space that the user would like to explore further [241]. Such user interaction not only speeds up the computation by focusing it on the data subsets of interest but also supports the user in gradually building a mental model of the data and forming hypotheses. PDA systems enabling this form of interaction have proven to be successful for exploration of data subsets in Twitter data [16] and patterns in a large set of medical event sequences that are progressively revealed by the computation [293]. Showing provenance information, i.e., meta-information about the simplifications made to the current intermediate results, is important in allowing the user to make more informed decisions about which parts of the data space is worth exploring and which could actually be pruned [207].

7.4.2 F2: Focus on the Algorithm Space

When focusing on the progress of the algorithm, analysts aim to improve the results of the progressive analysis.

The Observer (RO1) can **Monitor the Computation (U4)**, aiming first and foremost to gain an understanding of the inner workings of a computational process. To this end, the absolute progress of a process can be visualized sequentially and in real-time in the form of an algorithm animation [155]. In visual analytics, animations of complex network layouts and data clustering algorithms are routinely used to help users understand and tune their parameters. Similarly, it is possible to smoothly display the evolution of complex algorithms and to observe, for example, the inner workings of a neural network, such as the dataflow between input, hidden layers, and output [288].

The Searcher (RO2) can **Adjust Algorithm Run-time Parameters (U5)**, interrupting the computation, changing parameter settings, or swapping out entire computational modules or models. They can evaluate new algorithmic configurations by test-driving them on datasets with known ground truth in order to find one that satisfies the given computational objectives. In this case, the user can run multiple executions of the same progressive computation using different parameter sets, compare their intermediate results, and successively terminate executions once it becomes obvious that they are not performing according to the analyst's expectations [11]. A similar approach can also be adopted when debugging an algorithm to, for instance, compare the results of different rule execution sequences as the rules are dynamically executed [308]. **Research Agenda:** When the Searcher runs a black-box or complex model as during training of a deep neural network classifier, it is not trivial to identify the leading cause of the change in the progressive outcome in order to determine how to adjust parameters or which module must be corrected in priority [231]. Various explainable approaches and techniques [7, 264] could be explored to guide the Searcher in making these adjustments.

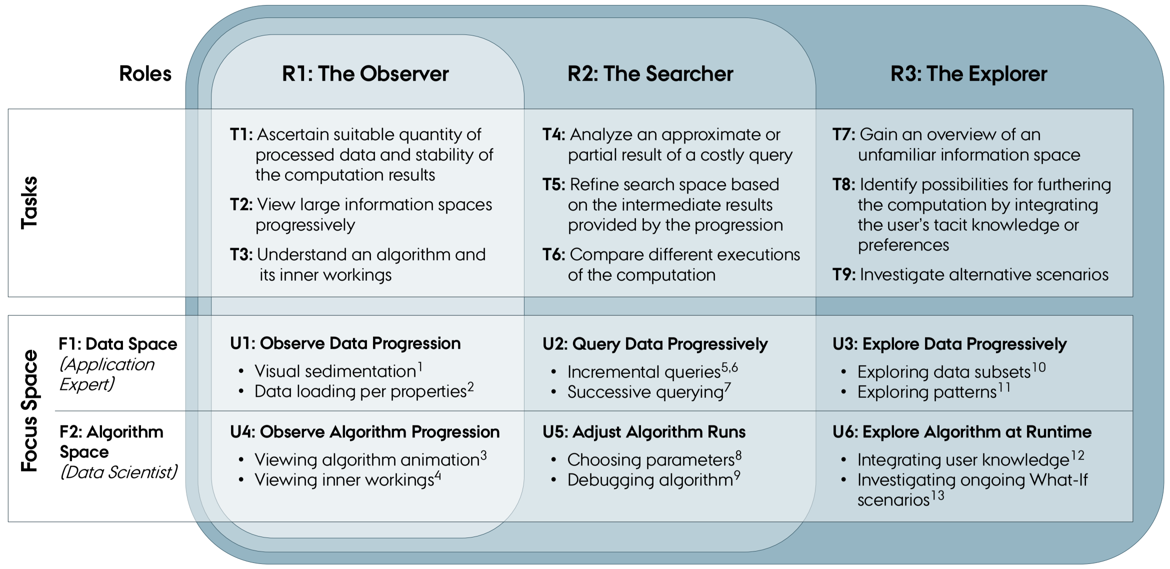
The Explorer (RO3) can **Explore the Algorithm at Run-Time (U6)**, that is, browsing through and experimenting with the multitude of process behaviors as they result from different algorithm choices or parameters. A PDA system can make this possible by allowing the user to not only view the inner workings of an algorithm but also interact with it at run-time to further explore performance or stability aspects of the algorithm, and also to steer it through user input. For instance, PDA can allow users to integrate tacit knowledge and preferences into a typical unsupervised dimensionality reduction or clustering algorithm (e.g., Self-Organizing Maps) as it runs, leading to improved cluster results [269]. In other cases, a user may want to explore the parameter space of a computation while it is running, as is the case in interactive flooding simulations where different constellations of water influx and barriers are tested in What-If scenarios [325].

7.5 User Complications and Biases

One of the main characteristics of PDA is its ability to increase user involvement in visual data analysis. The user introduces into the analysis process not only creativity, background knowledge, and past experiences, but also personal preferences and the subjectivity of human judgment. Unfortunately, this entails risks: the user may jump to conclusions based on highly unreliable intermediate results, or the possibility of steering away from a seemingly unpromising analysis path that would have yielded important insights later. In this section, we look at four different such pitfalls that complicate the use of PDA by the human analyst.

Using incomplete results to confirm a preferred hypothesis. This complication is particularly relevant for the Searcher, who works towards a clear analysis goal for which a desired or "most plausible" outcome may already been known upfront. Given the incomplete and often fluctuating nature of intermediate results in PDA, confirming a desired result can be seen as an exercise in finding the right time to stop the progressive computation. Note that such a misuse of PDA may not necessarily be a conscious act, but can also be a result of *confirmation bias* [188]. This well-known phenomenon captures an analyst's skew towards finding insights and drawing conclusions that are in line with prior-held personal beliefs. Even if the progression of intermediate results is mixed and also shows outputs that could be used to refute a hypothesis—human users tend not only to ignore evidence that rejects a believable conclusion (*Belief bias* [79]), but also to exaggerate the evidence supporting the preferred hypothesis (*Exaggerated expectation* [319]).

Research Agenda: PDA systems could try to mitigate the *confirmation bias* by



	Roles	R1: The Observer	R2: The Searcher	R3: The Explorer
Tasks		**T1:** Ascertain suitable quantity of processed data and stability of the computation results **T2:** View large information spaces progressively **T3:** Understand an algorithm and its inner workings	**T4:** Analyze an approximate or partial result of a costly query **T5:** Refine search space based on the intermediate results provided by the progression **T6:** Compare different executions of the computation	**T7:** Gain an overview of an unfamiliar information space **T8:** Identify possibilities for furthering the computation by integrating the user's tacit knowledge or preferences **T9:** Investigate alternative scenarios
Focus Space	**F1: Data Space** *(Application Expert)*	**U1: Observe Data Progression** • Visual sedimentation[1] • Data loading per properties[2]	**U2: Query Data Progressively** • Incremental queries[5,6] • Successive querying[7]	**U3: Explore Data Progressively** • Exploring data subsets[10] • Exploring patterns[11]
	F2: Algorithm Space *(Data Scientist)*	**U4: Observe Algorithm Progression** • Viewing algorithm animation[3] • Viewing inner workings[4]	**U5: Adjust Algorithm Runs** • Choosing parameters[8] • Debugging algorithm[9]	**U6: Explore Algorithm at Runtime** • Integrating user knowledge[12] • Investigating ongoing What-If scenarios[13]

Figure 7.1: User Roles (RO1-RO3), Tasks (T1–T10), Focus (F1–F2), and Usage (U1–U6) in Progressive Visual Analytics systems. Usage examples: [1] *et al.* 2013 [132], [2]Rosenbaum & Schumann 2009 [253], [3]Kerren & Stasko 2002 [155], [4]Smilkov *et al.* 2017 [288] [5]Hellerstein *et al.* 1999 [116], [6]Fisher *et al.* 2012 [89], [7]Khan *et al.* 2017 [156], [8]Angelini *et al.* 2018 [11], [9]Urban *et al.* 2003 [308], [10]Badam *et al.* 2017 [16], [11]Stolper *et al.* 2014 [293], [12]Schreck *et al.* 2009 [269], [13]Waser *et al.* 2010 [325].

supporting the user in carrying out an "analysis of competing hypotheses" [120] before making a decision. For instance, some analytic tools try to make it easy to link evidence confirming or rejecting different hypotheses to the analysis (e.g., [335]); others try to incorporate it as an integral part of the decision process [331]. Another approach is to ensure a certain degree of coverage of the full dataset by an intermediate result [322] before the option to terminate the progression becomes available to the user.

Read something into incomplete results that is not there. The risk of falling prey to *apophenia*—i.e., seeing relations between unrelated things—is already high in eager visualizations [163]. In PDA, that risk can be aggravated as false patterns and visual artifacts can appear in the stream of intermediate results due to the progression and not because they actually exist in the full dataset. This is particularly problematic if the analyst uses these patterns as insights to terminate the computation early and move on to the next analysis step or a decision based on them. There are multiple reasons why an analyst might do so:

- **Overconfidence**: the analyst (e.g., the Observer RO1) overestimates their understanding of the progressive process and its likely outcome (cf. *Overconfidence effect* [203, 204]).
- **Confirmation**: the observed patterns "confirm" a prior assumption held by the analyst (e.g., the Searcher RO2) (cf. *Confirmation bias* [188]);
- **Delay discounting**: the analyst (e.g., the Explorer RO3) values an insight gained now higher than the possibility of a different insight gained at an unknown later point (cf. *Present bias* [91, 219]);

It should also be noted that frequent exposure of the analyst to intermediate results that are not representative of the final result increases the chances of the analyst considering the illusion as true (*Illusory truth effect* [111, 197] and *Attentional bias* [139]). Similarly, priming and anchoring effects [309], according to which later judgments will be affected by earlier stimuli, might play a more pronounced role in progressive systems.

PDA systems could try to mitigate apophenia in several ways. One might be to encourage users to specify the assumptions they take away from an intermediate result in an explicit form—e.g., as progressive safeguards [141]. For example, a user searching the New York Taxi dataset [214] might specify their belief that "over half of daytime taxi rides go through this six-block region of midtown Manhattan"). This allows the system to continuously check these assumptions and to alert the user if any of them no longer hold. Of course, this only works if the computation is not terminated after gaining the insight on which the user's assumption is built. One way to achieve this is to continue the computation as a background process that notifies the user of any major changes and makes it possible to reverse any analysis steps completed since [11].

Waiting for information irrelevant to the intended goal. In PDA, analysts can easily get distracted by the constant influx of updated results, leading to a paralysis in their decision-making ability. This traces back to the *Information bias* [18], which leads users to believe that more information also leads to more informed decisions—even when this information is irrelevant to that decision. Interestingly, the opposite is often true, as a number of decision strategies and heuristics have actually been shown to work better with less information. This is known as the *less-is-more effect*, and it has been found, for example, when judging the more valuable of two objects [17]—apparently, multi-criteria optimization is not something that comes naturally to the human user.

For example, PDA systems could try to mitigate this complication by showing density plots that aggregate the data points and do not show them individually. This way, adding a few individual data points here or there will not create a visual cue suggesting that there is still a lot of new data with unseen properties worth waiting for. Instead, the color of the bins in which they are placed will simply turn a slightly darker shade. This strategy was, for example, applied by Badam *et al.*'s progressive tweet map [16].

Misjudging the uncertainty of intermediate results. The intermediate results shown by a PDA system are incomplete approximations based on only part of the data, as discussed in Chapter 6. Before making a decision based on these intermediate results, the user must judge the uncertainty of the depicted results. Yet if, for example, the progression exhibits fluctuations in the results and generates seemingly ambiguous outcomes, the user might not trust an otherwise good approximation.

Thus, it is important to use effective representations of probabilistic data in PDA systems. Yet, the way that uncertainty of the intermediate results is presented to the user will greatly influence any decisions made using this information. User behavior is likely to be different depending on whether the inaccuracy or accuracy of an intermediate result is shown, as humans react differently to the same information being presented negatively or positively. This is known as the *Framing effect* [306], which finds that humans make riskier decisions when being confronted with negatively framed information. For PDA, this could mean that users are more likely to steer the progression away from computations whose uncertainty is presented as inaccuracy—i.e., something undesirable with a negative connotation—than from the same computation whose uncertainty is presented as accuracy.

PDA systems can try to mitigate this effect by asking users for their rationale and logging it when, for example, terminating a progressive computation or changing its parameters. This would not only improve the reproducibility of the analysis, but it is also known to effectively counter the framing effect by preventing "gut decisions," which are prone to be affected by such biases [198, 299].

7.6 Application Example: Crisis Management

This section brings together the roles, tasks, and biases and discusses them in a practical example. Specifically, we discuss a usage scenario illustrating how the understanding of the human user outlined above can guide the design of PDA systems. We take the example of a designer in charge of improving a crisis management system.

7.6.1 The Background: Visual Analytics for Crisis Management and Emergency Response

During natural or man-made disasters, it is critical to get the relevant emergency response at the correct location and time. During such crises, people increasingly use social networks like Twitter to get the latest updates about the disaster event, post about their own situation, or report on possible damages or danger nearby them, asking for or offering help [40, 129, 291]. This online stream contains valuable information as text, pictures or video, which are very useful for humanitarian or governmental organizations to gain early insights from an on-going situation [134]. Beyond aftermath analysis [5], studies have shown the utility of this online content for disaster response and management [215, 257] if processed timely and effectively, but this task is very challenging and is the topic of ongoing research [64].

The Artificial Intelligence for Disaster Response (AIDR) system [135] processes streams of millions of social media data with automatic classifiers trained with crowd-sourced data tagged by volunteers. This system has been enriched with an interactive dashboard [14] (Figure 7.2) to visualize social media data during crisis events. Such decision support systems must facilitate collaborative decisions involving different types of users:

- the **Emergency Rescuer** (ER) leads a team of rescuers in the field, looks for guidance from and reports to the Crisis Responder;
- the **Crisis Responder** (CR) is the application expert able to interpret the meaning of the data and to decide about emergency actions and messages to send to the ERs;

- the **Data Analyst** (DA) is the visual and data analytic expert collaborating with the CR to support timely delivery of relevant automatic models, processed data, and visualizations.

These systems face multiple technical challenges. There are a huge number of messages posted on social media at any time and even more during crisis events, but only a small part of social media data is relevant to a given ongoing crisis event. Interactive monitoring tools can provide rapid access to the most critical and important situational information to support humanitarian organizations or affected people such as *persons*, *organizations* and *locations* [14].

These tools can be technically complex. All forms of media need to be processed: both images and text can contain critical information. Image processing has been used to extract actionable information (i.e., damage severity assessment) from social media images to get situational awareness during an ongoing crisis event [4, 220]. Natural Language Processing (NLP) techniques have been used to automate the filtering of text data like *reports of injured or dead people*, *urgent needs*, *missing persons*, *critical infrastructure damages* etc., but identifying this information is very difficult because many social media messages are short, informal, and use slang and abbreviations informal language [136, 215].

These models are prone to errors either due to uncertain labels gathered through crowdsourcing, inherent uncertainty until field checking, erroneous model prediction, or irrelevant feature spaces. Time is a scarce resource during a crisis, and can cost lives—but it takes time to train these models, but also to assess that a noisy pattern is actually a signal. Indeed, a processing bottleneck may be human effort: the AIDR system [135] (Figure 7.2) uses supervised classification trained through live crowd-sourcing with thousands of volunteers tagging messages and images *in real time* during a crisis event [220]. The CR decides **topic categories** of interest to which they want posts to be assigned by the volunteers, like *reports of injured or dead people*, *urgent needs*, *missing persons*, *critical infrastructure damages* etc. Volunteers tag posts; the posts are used to further train the classifier.

This example corresponds to a progressive scenario. Presuming that all the relevant social media messages have been ingested, there is a definite end result to reach for each event—its characteristics in the real world to guide relevant, timely emergency response. Time is bounded, but computation cost is not. As we extract images and run textual analysis on more postings, the intermediate result should get progressively closer to the end result by processing more and more data.

Note that here, we intentionally frame this as a PDA problem, not a streaming one. In reality, it is likely that additional social media posts might also be streaming in, adding a temporal component—meaning that the end result might also change dynamically.

We can now look at the roles we established earlier. The Data Analyst (DA) is likely to act as the *Explorer*. The DA is interested in understanding how the algorithms are performing, and may tune the image recognition and NLP algorithms as they learn more about the data. For example, certain scenes and keywords may recur in the postings, and it may be useful to train the system to recognize these keywords and identify them.

The Crisis Responder (CR) is likely to act more like the *Searcher*. They will be looking to answer specific questions: where resources need to be sent, what resources are needed, and where problems are emerging. They will be interested in focusing on the domain of the algorithms, asking the NLP to focus in particular on posts relevant to specific topics.

Finally, the Emergency Rescuer (ER) will look like the *Observer*. They are likely to be interested in the progress of the analysis so that they know when they will be dispatched and will want to see partial results to start their planning process. However, they are less likely to manipulate parameters or alter queries.

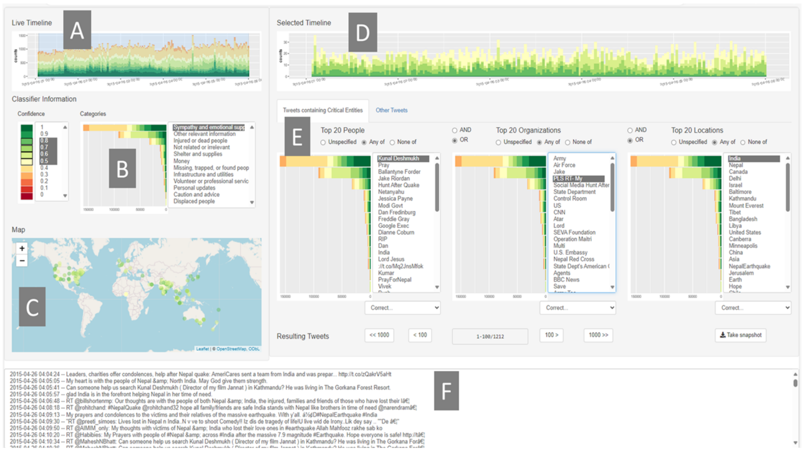
7.6.2 An Example Scenario: Coordinating Wildfire Response using incoming Twitter Data

We describe a usage scenario where the designer of PDA systems observed a crisis management team using the AIDR system during a dramatic event, and is tasked to provide this system with progressive functionalities. Here is the designer's report annotated with usages (Uj) and tasks (Ti) from Figure 7.1 and issues (ξk) also collected in Table 7.1:

A wildfire fueled by strong winds has raged for two days in California and is currently intensifying. The fire has reached villas and residences near Malibay County and is approaching Las Esmeralda's densely populated suburbs.

The CR collects information from ERs in the field and from social media. Automatic processes extract data and images from tweets based on location, text, and image content. The DA sets the model's parameters based on previous wildfire and hurricane historical tweet records, and from the last two days of crowd-sourced image and text tagging.

At some point, the name "Brad Spears" starts appearing in the top-K persons from messages in the category "critical infrastructure damage". The CR looks at the related tweets. They turn out to be from press coverage, telling how the hurricane



```
2015-04-26 04:04:24 -- Leaders, charities offer condolences, help after Nepal quake: AmeriCares sent a team from India and was prepar... http://t.co/zQakrV5aHt
2015-04-26 04:05:05 -- My heart is with the people of Nepal &amp; North India. May God give them strength.
2015-04-26 04:05:41 -- Can someone help us search Kunal Deshmukh ( Director of my film Jannat ) in Kathmandu? He was living in The Gorkana Forest Resort.
2015-04-26 04:05:57 -- glad India is in the forefront helping Nepal in her time of need.
2015-04-26 04:06:48 -- RT @billshortenmp: Our thoughts are with the people of both Nepal &amp; India, the injured, families and friends of those who have lost their lâ€¦
2015-04-26 04:08:18 -- RT @rohitchand: #NepalQuake @rohitchand32 hope all family/friends are safe India stands with Nepal like brothers in time of need @narendramâ€¦
2015-04-26 04:09:13 -- My prayers and condolences to the victims and their relatives of the massive earthquake. With y'all. á½¢D#NepalEarthquake #India
2015-04-26 04:09:30 -- "RT @preeti_simoes: Lives lost in Nepal n India..N v ve to shoot Comedy!! Iz dis de tragedy of life!U live wid de Irony..Lik dey say .. ""De â€¦"
2015-04-26 04:09:50 -- RT @AIMIM_only: My thoughts with victims of Nepal &amp; India who lost their love ones in #earthquake Allah Mahfooz rakhe sab ko
2015-04-26 04:10:20 -- RT @Habibies: My Prayers with people of #Nepal &amp; across #India after the massive 7.9 magnitude #Earthquake. Hope everyone is safe! http://tâ€¦
2015-04-26 04:10:34 -- RT @MaheshNBhatt: Can someone help us search Kunal Deshmukh ( Director of my film Jannat ) in Kathmandu? He was living in The Gorkana Forâ€¦
```

Figure 7.2: The AIDR dashboard prototype [14] has only basic progressive functions (see Table 7.1). It allows for filtering tweets by time (A,D), categories with color-coded confidence level (B), and location (C). The numbers and names of top-K critical entities (persons, locations, organizations) are displayed as bar charts (E), which can be selected and filtered to show only the most relevant messages (F).

ripped off the roof of his Malibay villa. A local ER team is sent on-site (ξ1). The CR takes note of this name and manually filters it out of the displayed results (ξ2). The CR focuses their attention on a collapsed bridge that appears as a likely event in the filtered categories and in the related tweets they read based on the increasing number of occurrences (U2-T4). If the bridge is damaged, it could prevent emergency vehicles from reaching a strategic area to fight the fire. They decide to query for related confounding locations with the same name mentioned in the corresponding messages and are supported with the detailed map of related geo-located tweets (ξ3). Classifier confidence indicators (U1-T1) help the CR decide to redirect a few ERs toward the two most likely locations for further verification. They get a trustworthy detailed report from the second one: "The Linden bridge is damaged but not collapsed. But a multi-story residential building is on fire with about 10 people on the rooftop, and the nearby woods are already on fire." The CR collects information from other ER teams to redirect a squad there and lets the ER leader organize on-site tasks.

Meanwhile, the DA is checking ongoing tweet categorization and NLP techniques, when he spots slow convergence of the training process of the model (U4-T1). Looking at the current errors on the validation set, some labels look buggy (U4-T3) (ξ4). They send new requests on social media to recruit more crowd-sourced tagging volunteers in the hope of increasing filters' accuracy.

7.6.3 The Design Guidance: Identifying Progressive User Requirements for such a Scenario

Analyzing their notes, the designer determined that the CR and DA have a Searcher role (RO2) based on the type of tasks they can achieve with the current system. The ERs, however, are not informed based on a progressive approach. They get their orders from the CR decision without indication of potential uncertainty and alternative options that led to it (Table 7.1(ξ1)). ERs could benefit from being part of the PDA system as Observers (RO1): progressively updated by the system with the certainty degree of concurrent options (building/bridge/critical asset location, severity of the damages) typical of (U1-T1) task, combined with data they would collect in the field (physical obstacles, local risk conditions, asking more relevant questions to local eyewitnesses, etc.), the ER team leader could take a better-informed decision on how to manage their team on the spot and provide a faster and more accurate emergency response.

Research Agenda: This design process rests upon matching the user tasks (Section 7.3(T1-T9)) with the standard typology of visualization tasks [35], creating opportunities for developing a new typology and taxonomy of visualization tasks and idioms that would better support the graphic and interaction design specific to PDA systems.

The CR would like to black-list messages related to Brad Spears' villa, but current controls only hide irrelevant messages containing this entity with no effect on the actual filtering process (Table 7.1(ξ2)), so the incoming message can still contain messages related to that irrelevant event. A progressive system would modify the filters to make space for more relevant critical entities among the top-K (U2-T5) and steer the filtering processes (U6-T9) in future collected messages as well. The map shows the geo-location of the sender of the tweet message, which may not be related to the one of the event itself (Table 7.1(ξ3)). The CR would benefit from seeing a progressive location of the events (U1-T2) with decreasing uncertainty based on an analysis of the contents referring to the same event using geo-coding and NLP techniques. The CR is otherwise using the AIDR system in a progressive way

Users	Action/Event	AIDR	Proposed PDA functions
ER	Move to the event location	Get decisions from CR (ξ1) Lack of context leading to non optimal decision	Get certainty level of concurrent options (U1-T1)
CR	Brad Spears' villa is damaged	Filter on the display (ξ2) No impact on back-end filtering processes and classifiers	Refine filters (U2-T5) ◆ Steer ongoing processes (U6-T8)
	A bridge collapsed	Classifier confidence indicators (U1-T1) ◆ Geo-located tweets on a map (ξ3) Not related to the event itself ◆ Analyze partial results (U2-T4)	Event progressive location on a map (U1-T2)
	Exploring hypotheses	*Not supported*	*Exploration role (U3-T8,9,10)* (ξ5) Not during the crisis, but relevant in the crisis aftermath
DA	Monitoring classification and NLP processes	Spot slow convergence (U4-T1) ◆ Identify low accuracy (U4-T3) (ξ4) Lack of control and exploration capacities	Analyze parameters at play (U5-T4) ◆ Refine models (U5-T5) ◆ Compare different executions of the models (U5-T6) ◆ Control of the model features (U6-T9) ◆ Investigate alternative scenarios (U6-T9)

Table 7.1: Comparison of the current AIDR system and its improvement, adding progressive functionalities (Section 7.6.3).

(U1-T1,U2-T4). Still, the feedback (black-listing) and decisions are just annotated on maps and visual indicators, but not used to steer the ongoing processes (U6-T9).

The designer also observed that the DA plays a crucial role in an emergency situation to support the CR. The current interface partially supports the DA in a Searcher role (U4-T1,T3), but the DA's expertise is not fully exploited (Table 7.1(ξ4)). The system could support them to analyze parameters at play (U5-T4), refine models (U5-T5), and compare different executions of the models (U5-T6). The DA could also take an Explorer role (RO2) to have more control over training features (U6-T9) and test hypotheses (U6-T9) during the training of the classifiers so as to get the most out of the tools at the earliest time.

The designer finally assumes that the CR could also benefit from exploring data progressively (U3-T8,9,10).

When inquiring about these potential unforeseen user needs, ERs and DA were interested in furthering these options. However, the CR mentioned that having possibilities to test different scenarios (T9) as part of the Explorer role is not crucial during the unfolding of a crisis where the focus is on real-time rescue (Table 7.1(ξ5)), but is important to have in the aftermath of an event to better organize future missions. So the CR does not actually need a progressive approach as an Explorer.

As we see in this example, the designer would use our framework to inform their solution and ask questions about the type of support that could be helpful to the users at different times of their missions, actually useful needs they might not have expressed directly. Beyond the initial exploration and collection of the users' needs, they could refer to the existing literature related to the specific usages (Notes in Figure 7.1) to inspire and inform their technical design and propose the best solution to support these progressive tasks for their respective users.

The pitfalls described in Section 7.5 will also be taken into consideration by the designer and could be discussed during a collaborative design process [337] with the users to determine the actual risks of such biases, their impact, and ways to mitigate them.

7.7 Summary

We have laid out a space for thinking about user tasks for progressive analytics. The three user roles—passive Observers, data-based Searchers, and active Explorers—suggest not just different tasks, but different designs. We believe that this distinction lays out a space for future design work that further refines and adapts these different roles to different applications.

In identifying the work done in PDA, we have added another type of analysis loop to the traditional visual analytics loops: where we might have once queried, responded, and refined, now there are new sub-loops around acting on partial results, verifying those results, and steering computation. **Research Agenda:** Just as we have rich knowledge of the requirements around query systems that can produce interactive answers, we now need to learn what the appropriate requirements are for PDA systems. What are the trade-offs between interactivity, accuracy, and timing? When is it worthwhile to produce more refined output, compared to producing responses to users? What sorts of users will tolerate a PDA system that takes longer than a baseline system to reach a final response—but can give partial results much more quickly?

The space of PDA design and understanding how users can take advantage of it can broaden across different areas of visual analytics. Can we take advantage of PDA techniques to design dashboards, for example, that make use of smaller sets of data, or are based on sampled results? How can we take advantage of our insights about bias and uncertainty to improve our ability to create traditional baseline chart types? And how can we leverage and amplify the positive effects that active steering brings with it—e.g., the self-generation effect [65] that indicates that self-generated information can be recalled better than information that is simply read, or the IKEA effect [217] that captures that humans place a higher value on outcomes they (partially) created themselves.

PDA offers amazing opportunities for empowering users in new ways to explore and understand datasets that previously could not have been analyzed in a visual-interactive way. This chapter provides a framework for understanding user needs and building tools that support them.

The attentive reader will by now have noticed that this chapter only speaks of the user in their singular form. **Research Agenda:** Collaborative PDA scenarios involving multiple users remain largely unexplored. While data prioritization and computational steering based on user interaction/interest works well in single-user settings, it is still an open question as to how to realize similar mechanisms for multiple users with different user interests, possibly even different user roles. Can this be achieved with a joint visualization, or should they each have their own display giving them their own tailored view on a joint progression? Or does each user even run their own tailored progression in the background? These questions are open to further investigation in the future.

8 Progressive Methods in Machine Learning

8.1 Introduction 132
8.2 Motivation 134
8.3 Progressive Machine Learning Algorithms 135
8.3.1 Characteristics of ML Algorithms 135
8.4 Examples of Progressive ML Algorithms 138
8.4.1 *k*-Nearest Neighbors Classifier 138
8.4.2 Support Vector Machines as Classifiers 140
8.4.3 Deep Neural Networks 143
8.5 Challenges and Recent Advances 145
8.6 Summary 147

Authors Cagatay Turkay, Nicola Pezzotti, Hendrik Strobelt, and Barbara Hammer

8.1 Introduction

Having looked at how to carry out Progressive Data Analysis, we now turn to a particular application area that might specifically benefit from progressive data analysis approaches: the field of Machine learning (ML). ML is now the major driving force behind many successful Artificial Intelligence (AI) applications. These applications often involve models that take a considerable amount of skill, creativity, and time to design and train due to the vast space of parameters and data features that need to be decided on during the design of the model, and due to the large volume of the data sets used in their training [324]. This process often involves several iterations and trials until satisfactory outcomes are reached [305]. This laborious model building process is then followed by the model being used in the wild by various stakeholders who need to understand how the model is performing, in particular when faced with the dynamically evolving characteristics of the "real world". Progressive methods offer significant improvements and enhancements to these different processes. Progressivity can improve the quality of models by empowering the model builders, and support end-users of models in making decisions while ensuring that the models continue to function over time, hence they are not subject to any distribution drift.

With the ubiquity of such AI systems, different users interact with them at various stages and mapping these users is an important first step in thinking about the role of progressivity in these processes. Inspired by the user roles in Section 7.2, we can classify ML users into three groups: *Architects*, *Trainers* and *Model Users*. These three ML roles and the different contexts they appear in are illustrated in Figure 8.1. These roles should be seen as archetypes, and thus the same person can cover multiple roles at once.

Most AI applications start with the definition of a task, e.g., classification of images among a number of labels, and a large amount of data is collected that represents this task. A model is defined by a machine learning *Architect*; a researcher that develops new models to solve the defined task. The model is then trained to perform well on the task given the exemplary data, also called the training set. The user responsible for training the model is the *Trainer*; the person that will take a machine learning model and optimize its trainable parameters to achieve good results on the task at hand. In this case, the Trainer defines the objective function for which the model will be optimized, e.g., based on the misclassification rate of the model. The Architect and the Trainer can be the same person. As a matter of fact, this is common during the development of new machine learning models.

Finally, once the model is trained, it is deployed to perform the task in an applied setting, which we refer to as *model deployment*. Once a model is deployed, a *Model User* will interact with the model. Different stakeholders for a given model can be seen as *Model Users*, even if they do not interact directly with it; similarly, a *Model User* may not be aware of the fact that they are interacting with a model. For example, machine learning recommender systems are employed in the many web platforms such as YouTube, Netflix, and Amazon. The Model User can interact with the model in a passive way simply by receiving the recommendations, or in an active way by steering the model according to their preferences. Progressivity is important at all stages, although its benefits may vary.

When linking these three roles to the ones depicted in Chapter 7, an *Architect* would largely be an ***R3**: Progressive Explorer* (with their focus on the algorithm space, **F2**). A *Trainer* would likely take on the role of the ***R2**: Progressive Searcher* (with a focus on the data space **F1** & as well as the algorithm **F2**).

In a progressive ML system, the *Model User* would be an ***R1**: The Progressive Observer*. It is important to note that in certain cases, the model user may not be aware that they are interacting with a progressive model since they might be only

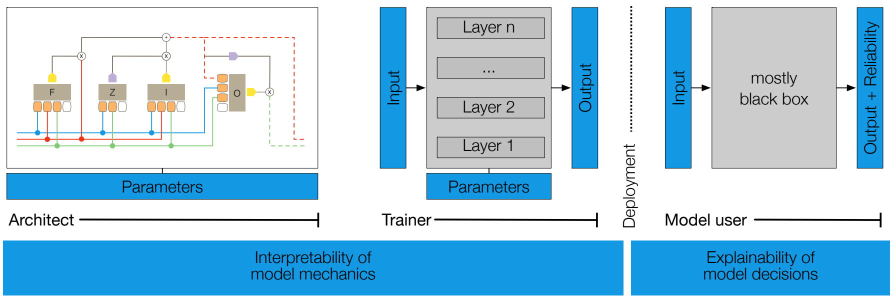


Figure 8.1: Different users and expectations for progressive machine learning

interested in outcomes, e.g., a streaming service user getting recommendations.

8.2 Motivation

Applying machine learning solutions in complex applications requires understanding the impact of the choice of chosen algorithms, their parameters and hyperparameters. This is a challenging task and an increasing research effort is spent on Explainable and Interpretable AI. However, the *Architect* or a *Trainer* usually must wait until a new model is successfully trained before they can begin to assess and interpret the model. This is a lengthy process that reduces the effectiveness of the development of new machine learning solutions. Similarly, a Model User might also want to observe several versions of a model by varying parameters that could be interactively manipulated based on the tacit domain knowledge they might have. Progressive methods would foster such interactive exploration.

Modern models require weeks to be trained on tens, if not hundreds of GPUs that are deployed in expensive cloud environments. A good motivation for the development of progressive machine learning algorithms can be found in the GPT-3 model from OpenAI [37]. It is the state-of-the-art Natural Language Processing model which consists of 175 Billion parameters. It was trained in days and at a cost estimated at over 10 Million dollars. However, the authors report that a bug in the implementation was found only after training, introducing a contamination of the test set. Due to the cost of training, it was infeasible for the authors to fix the problem.

Progressive machine learning methods would compute the results in a progressive manner, returning a first approximation of the results quickly with iterative refinements until converging to the final result [207]. Enabling Architects and Trainers to interact with intermediate results allows early detection of buggy implementations, or suboptimal parameter choices. Architects could quickly refine algorithmic features, parameter choices, and even the underlying machine learning technology. Model Users might benefit from being able to interact with many alternative models and outcomes, allowing for a more in-depth understanding of the implications of the outcomes. Enabling a progressive computation and verification of the model could potentially make these sorts of issues, as reported by OpenAI in the GPT-3 model, less frequent.

8.3 Progressive Machine Learning Algorithms

In this section, we elaborate on those characteristics of ML methods that might make them amenable to progressivity, and discuss their relevance for the different users. Like other algorithms, a progressive ML system will need to be designed to have several attributes that are not unlike those we have seen in Chapter 4. Most notably, the ML algorithm must exhibit both *convergence* and *stability* with respect to the resulting quality of the progressive methods. Optimally, the ML algorithms should exhibit *steerability*, allowing the user to interact with the system during processing to influence the methods and parameter choices, as well as the training set or features used, or even to interactively specify external logical constraints.

We will examine the progressive nature and steerability of several standard ML algorithms, such as kNN, SVM, and Deep Learning, and show that developing progressive variants involves a diverse set of challenges related to the areas of data management, ML algorithms, and visualization & interactivity.

8.3.1 Characteristics of ML Algorithms

Certain properties are required of Machine Learning algorithms in progressive settings. First, they must provide guarantees on their convergence (see Section 2.2) to a locally optimal solution. While this property is intrinsically required by all ML algorithms, a further required property of the convergence is that it must be stable (see Section 2.2). Therefore, the performance of the model should gradually improve with respect to its objective function with more data, more iterations, or both. This is an important requirement for the application of ML algorithms in a progressive setting; if the algorithm is not stable, the *Architect* and the *Trainer* observing the partial results will not be able to trust them should they fluctuate wildly over time. Note that, while we require the algorithms to converge, we do not expect monotonicity for the objective function that is optimized. In the following, we build on the characteristics of progressive algorithms outlined in Chapter 2 and present more specific criteria and characteristics for progressive machine learning methods.

We identified the following criteria one would look for to assess the progressive nature of an algorithm:

- **Ability to Work on Partial Data**: The algorithm processes and learns from large datasets. Intermediate models are obtained by taking into consideration only a subset of the total data. An example of this kind of progressivity can be seen in kNN classifiers [90], where the k most similar items to the query at hand are searched in a large dataset.
- **Iterative Computation**: The algorithm is iterative in nature and intermediate results or models are generated at every iteration. An example of this type of progressivity is the k-means clustering algorithm [186] where, in each iteration, the centroids identifying the clusters are moved according to the data distribution (see the example later in this chapter).
- **Increasing Quality due to Heuristics or Approximations**: Algorithms may rely on heuristics and/or approximations. The quality of the approximation can be progressively improved while the algorithm evolves. Examples of this type of progressivity are manifold-learning algorithms that rely on approximated k-Nearest Neighbors queries [192, 232, 233].
- **Evolution over Time**: Progressive computations are closely related to the notion that results improve over time. However, for some techniques, time is the only means available for characterizing their progressivity. Continual learning techniques [182, 279], i.e., machine learning algorithms that evolve without predefined datasets to learn from, are an example of these techniques.

Note that ML algorithms may belong to more than one category. An example is the Stochastic Gradient Descent (SGD) algorithm [31] which is used in several contexts, including the training of deep learning models [170]. At each iteration of the optimization, samples that are randomly selected from the data are used to update model parameters. However, the iterative nature of the algorithm is complemented by the notion of the coverage of the dataset. In SGD, the samples are selected so that the dataset is fully covered in a number of iterations that are defined as epochs of training. Multiple epochs are then needed to converge to the local optimum.

Another key feature of progressive ML algorithms is their **interactivity** and **steerability**. We define the interactivity of an algorithm as the ability of a user to control it during its progressive computation by, for example, changing its hyperparameters.

We define *steerability* as the ability to control the algorithm in terms of the local minimum to which it converges.

We identified the following characteristics of an algorithm that enables its interactivity or steerability:

- **Algorithm Selection**: Different algorithms can be selected to train the same machine learning model. For example, several training algorithms have been developed for random-forest classifiers [36, 122]. The user can interact with the model by interactively changing the algorithm used for training.
- **Hyperparameters**: Hyperparameters are crucial for good performance and model generalization. While hyperparameter-tuning techniques can be adopted, as in AutoML [112], a user-in-the-loop approach may allow for a more efficient and effective selection [63].
- **Considered Features**: The user can perform feature selection and engineering and observe, in a progressive fashion, how the choices affected the algorithm at hand.
- **Logical Constraints**: Upon analysis of the data, the user can impose some logical constraints on the data. For example, features can be ignored for sub-parts of the data where they are not informative enough based on prior knowledge of the problem.
- **Model Complexity**: The user interactively controls the model complexity. As an example, the number of neighbors considered by a kNN classifier can be interactively changed and fine tuned by the user to achieve better results given the data at hand.
- **Training Set**: The user can add or remove data points from the training set to control the model performance. For example, the user may identify mislabeled points from intermediate models, which are then removed from the training set.

Note that while these are desired characteristics for an algorithm in a progressive computational environment, they can be difficult to achieve. Moreover, in cases where interactivity and steerability is maintained, there are questions as to their utility. One may argue that steerability is not needed if the learning is performed correctly and according to user needs, but how can one be sure that data is not corrupted and that a method chosen initially will perform as intended? **Research Agenda:** More research is required on this specific aspect of progressive computations for machine learning algorithms.

8.4 Examples of Progressive ML Algorithms

In this section, we elaborate on the characteristics above by introducing three machine learning algorithms and explaining how well they support progressive computations in terms of stable convergence, interactivity, and steerability. We have selected algorithms characterized by different degrees of model complexity with the aim of providing an overview of the potential of progressive computations in a wide range of applications.

As a simple example, we have selected a k-Nearest Neighbors classifier [90]. This model does not have a training phase, which is replaced by a look-up strategy in the existing training set. Therefore, progressive computations are adopted in this look-up, where several existing methods can be applied. As an intermediate example, we adopted support vector machines (SVM) [61]. SVMs include a training phase, which can be performed progressively. Moreover, compared to a kNN classifier, the training of SVMs requires the selection of a training algorithm and its hyperparameters. Finally, as our most complex example, we introduce Deep Learning Models, a set of techniques that in recent years achieved impressive results in several machine learning tasks [22]. Deep neural networks are complex models with a large number of hyperparameters. Moreover, the training of deep learning models is very slow, thus posing challenges in the progressive visualization of their training.

Research Agenda: Progressive methods need to be rethought to support the monitoring of the training of very large deep neural networks that are considerably slower to train.

8.4.1 *k*-Nearest Neighbors Classifier

As a promising method briefly touched upon in Section 4.4.4, we present the k-Nearest Neighbors classifier (kNN-classifier) as a first example of a machine learning algorithm that can benefit from a progressive computational framework. A kNN-classifier is a non-parametric model that was developed by Fix and Hodges in 1951 [90].

Given a training set $(\mathbf{x}_1, y_1) \ldots (\mathbf{x}_N, y_N)$, where N is the number of training items, $\mathbf{x}_i \in \mathbb{R}^d$ is the i-th training point with values in d dimensions and a corresponding class y_i, the goal of the classifier, given a previously unseen data point $\bar{\mathbf{x}}$, is to obtain the corresponding predicted label $\bar{y}$. The algorithm searches in the training set for the k nearest neighbors to the point $\bar{\mathbf{x}}$, where the Euclidean distance is often used as the metric. Once the k points are collected, the prediction $\bar{y}$ is obtained by assigning the label that is most frequent among the neighbors. The user has control of the prediction power of the algorithm in two ways. Firstly, the bigger and more representative the training set, the easier it is to find a close match for the point $\bar{\mathbf{x}}$, thus resulting in a more accurate prediction. Secondly, by changing the number k of neighbors used, the user can control the effect of the noise on the prediction. In the extreme case where $k = 1$, the noise in the training distribution, as well as the presence of outliers, may become dominant, thus resulting in a noisy prediction. Increasing k leads to a so-called smoother classification boundary, which has the effect of making the classifier more stable. However, this may come at the cost of misclassification, as the decision boundary will be directly effected, an effect that may be extreme when small sub-population of data are considered, i.e., when the size of the sub-population is a fraction of the selected value k. Therefore, the best choice of k depends on several factors, such as the size N of the training data, the number of sub-populations and the required trade-off between precision and recall of the resulting classifier.

In addition to its simplicity, the algorithm is also widely-used because of its easily interpretable results and superior performance when dealing with relatively low-dimensional input data. The main drawback with it has to do with the fact that a kNN query has to be performed for each data point that has to be classified. Compared with more advanced algorithms like SVM or deep learning-based classifiers, this can lead to high computation runtime caused by searching the neighbors. In its simplest form, the search can be executed by searching through all the available data points. This results in a computational complexity that is linear to the size of the data, hence $O(N)$ in big-O notation [273]. The computational complexity of the queries can be improved by adopting advanced algorithms and data structures. For example, by adopting tree structures the computational complexity of the search can be reduced to $O(\log(N))$. Examples of these algorithms are the KD-tree [93] or some of its extensions, which partition the space $\mathbb{R}^d$ into a hierarchy of regions. These regions are then progressively explored to identify the nearest neighbors. It is observed that, when the dimensionality d of the data is high, the performance of the algorithms degrades, becoming even worse than a linear search in the data, an effect know as the *curse of dimensionality*. To counteract this phenomenon, approximated searches can be used, where the trees are explored through heuristic criteria that provide the neighbors with a controllable approximation. An example of this approach is the FLANN algorithm [208]. Another widely used approach is known as locality-sensitive hashing (LSH). Here, similar points are placed together in the same *"bucket"* using appropriate hashing functions. The algorithm has a constant complexity $O(1)$, but it is an approximation by nature. Given the relevance of providing efficient kNN search at scale, several libraries have been developed over the years, with FAISS being one of the most notable examples [146].

Given the above description of the algorithm, it should not be surprising that the algorithm has great potential for a progressive computational framework. The key insight here is that the classification can be performed on an efficient and approximate computation of the closest neighbors, an approach that is often used in computer vision [256, 261] or in other machine learning tasks like dimensionality reduction [233]. In case of a suboptimal selection of the neighbors, the prediction will be less accurate and more noisy. However, given the fact that the prediction is easily computed from the neighborhood, the algorithm naturally allows for an improvement of the classification results by progressively improving the set of neighbors chosen. For example, a large set of the training set can be considered when the neighbors are collected. In this case, a progressively larger chunk of the data can be queried, leading to better results. More advanced solutions could make it possible to leverage the data structure used to query the data. All tree algorithms naturally support this, as the neighbors are located increasingly closer to the point $\bar{\mathbf{x}}$ as the query progresses. Seminal work on providing a progressive implementation of the kNN-classifier is PANENE, a progressive algorithm for indexing and querying approximated k-Nearest Neighbors [142]. PANENE allows updating of the kNN algorithm in a time bounded manner for an interactive representation of the results.

Research Agenda: A number of fundamental algorithms, beyond kNN search, may be reformulated as progressive computation and become building blocks of novel progressive ML systems.

8.4.2 Support Vector Machines as Classifiers

Support vector machines (SVMs) as proposed by Vapnik constitute particularly powerful machine learning classifiers, combining the principle of error minimization with structural risk minimization while training [61]. Unlike kNN classifiers, they do not lend themselves to an immediate extension toward an incremental learning scheme, and different proposals exist on how to extend classical versions to progressive variants [70, 258], using different principles, as elaborated in the following. SVMs implement nonlinear models which can be shaped to approximate any reasonable underlying function, provided a suitable choice of their parameters and hyperparameters [292]. Their modeling and the derivative of their training algorithm relies on the duality principles of constraint optimization [33]. Hence they offer both, a parametric and a non-parametric point of view: in its primal formulation, SVMs constitute parametric linear classifiers in kernel space. That means, they are essentially parameterized by a weight vector for a linear mapping, which is used to classify data. Data are pre-processed by means of a fixed nonlinear function: the so-called feature map which is associated with a kernel. This principle ensures that the resulting overall function is nonlinear in the data space and that universal approximation capabilities result.

Depending on the chosen kernel, however, an explicit representation in the kernel space can require a high (even infinite) dimensional feature space, and an explicit computation of the nonlinear pre-processing might be computationally demanding or even impossible. Therefore, an SVM is commonly phrased via its dual formulation: this provides an expansion of the functional form in terms of support vectors, which are part of the training set, hence this yields a non-parametric representation. The complexity of the dual problem and also the representation of the SVM prescription in dual form typically depends on the size of the data set. Rather than the feature representation, training in dual formulation makes use of the Gram matrix which is associated to the nonlinear feature map and is evaluated on pairs of the given data [271]. Since the dual training problem is convex, convergence to a unique global optimum of the cost function can be guaranteed—hence an iteration of training improves the quality of the result. Furthermore, strong learning theoretical guarantees ensure that an SVM, when trained on ever larger subsets of the training data, provides ever better approximations of the true underlying regularity [61]. Thus, a progressive treatment of SVM by stratifying along the number of iterations of SVM training or along the number of used training samples is possible [70, 258].

Yet, the classical formulation of SVMs hardly lends itself to learning for large data sets or efficient progressive analysis for larger data sets: the Gram matrix scales quadratically with the number of given training data. Hence training quickly becomes infeasible both in terms of time and memory constraints in the original formulation of Vapnik. Quite a few approaches have been proposed to overcome this challenge in machine learning, and they constitute interesting starting points for efficient progressive computations. Iterative SVM training processes data in chunks, whereby it heavily relies on the special property of SVMs to represent models in terms of support vectors rather than the full training data; this property enables an intelligent choice of suitable working sets, which guarantees a decrease in the training error over the process of training. These approaches combine convergence of the resulting quality with efficient incremental training [145], enabling efficient steering of the training set and resulting training time, respectively. Another approach approximates the training objective, such that finite core sets can be identified which steer the training, allowing computation time to be traded for the approximation quality of the optima in this approach [302].

A different line of approximation techniques controls the effort required to compute the full Gram matrix, which size scales quadratically with the number of training data: the so-called Nyström technique offers an efficient low-rank approximation in the dual kernel space [284]. This enables efficient incremental learning schemes in the dual SVM formulation [336]. As an alternative, efficient approximations of the data representations in the kernel space have been proposed such as random Fourier features, which are based on an efficient low-rank kernel expansion, or, more generally, efficient approximations of the nonlinear data pre-processing in the feature space [15, 99, 344]. Such approaches enable a direct training of SVM models on an approximate embedding in the nonlinear feature space by means of a parametric linear mapping. Since linear maps can easily be adapted incrementally, this treatment enables the direct use of incremental linear learning schemes [301]. Yet, while these approximations immediately enable an efficient progression along an increasing number of training samples, it is not clear how a refinement of the approximation can be implemented efficiently for both the Nyström approximation (regarding the choice of data subset at the core of the approximation) and random Fourier features (the Fourier components for approximation). A change or refinement would require an exchange of the data representation or kernel matrix, respectively.

In general, model hyperparameters of batch learning can become model parameters in incremental learning schemes as soon as the complexity and characteristics of the observed data can no longer be reliably estimated based on just a small data sample. SVM training requires the choice of a few hyperparameters: one important hyperparameter controls the trade-off between the training set accuracy and generalization ability of the training objective. Another one, which also controls the smoothness of the function, is the kernel width in the popular RBF kernel. While these parameters are crucial, they can be reliably estimated on smaller parts of the training data when those are representative of the data characteristic, such that their change does not seem necessary for progressive computations. Another crucial aspect, however, is the representation of the data, i.e., the choice of the kernel itself. Provided the data representation is changed, a new kernel representation results, i.e., the question of how to set up such operations so that an efficient reuse of previous computational results is possible in progressive learning remains a challenge.

SVM is typically considered a black-box technique, such that a control of its parameters and hyperparameters widely remains subject of trial and error. Few approaches exist that offer an intuitive integration of expert knowledge into SVMs:

due to its intuitive primal formulation and training objective, it is comparably easy to account for data set biases in SVMs, or to add additional linear constraints or error weighting to the problem formulation. Further, there do exist methods that enable the incorporation of invariances such as rotation or translation invariance into the shape of the kernel function [98]. Such technologies constitute an excellent starting point for the definition of intuitive interaction schemes which enable the user to progressively steer the learning process. Yet, while some constraints can be integrated efficiently, others, such as the choice of the kernel, currently require the complete retraining of the models. Interestingly, this is currently also required for a seemingly simple operation: deleting examples from the training set. While incremental learning is possible for SVMs, it is less clear how to forget parts of the training set if they are progressively dubbed irrelevant by the user.

8.4.3 Deep Neural Networks

Deep neural networks (DNN) are multilayered artificial neural networks. Each DNN model is based on a network architecture that commonly describes a nonlinear combination of neurons and layers of neurons [22]. Many types of neurons have parameters, e.g., weights and biases, that are learned during model training. Training is most commonly achieved by using a large amount of data presented to a version of back-propagation [171]. The goal of the training process is to optimize an objective function that describes the task. The training process itself is configured by hyperparameters. After training, the model and its optimized parameters can be used to infer output data for new input data with regard to the model task.

The training process of a single DNN is often time-consuming and requires significant computing resources, memory, and storage. It is inherently progressive because after each batch of training data has been processed, the model and its weights have an updated state. This state can be evaluated and compared to the previous state. In practice, this update evaluation is only done after some significant data has been processed, e.g., many data rows or all training data (one epoch). In many cases, the evaluation includes quantitative measures like loss value or performance on test data. Ideally, these measures should show monotonic behavior during the progression of the training process. In practice, these measures can vary and only show trends. Qualitative comparison of model training progression appears to be an understudied field and should gain more interest.

Research Agenda: The integration of PDA into deep-learning frameworks, such as Torch or Keras, is an open problem and will enable advanced applications.

Training a single model can help in finding a good parameterization for a predefined architecture and pre-specified hyperparameters. But to optimize for a particular task, searching for the right architecture and hyperparameters is itself a field of research that can be described as a progressive approach. In this case, each progression step is very costly and can consist of the full training of a model for the sampled combination of architecture and hyperparameters. Common automatic methods decide on the improvement of quantitative measures to either continue training a specific model configuration or cancel it. More sophisticated algorithms learn about favorable architectures and hyperparameters longitudinally across multiple tasks. Model search and hyperparameter optimization are often not monotonic or deterministic, i.e., not strictly progressive. We still think that user interaction paradigms for progressive search can be applied to this process.

The inference process should be quick and consume minimal resources. For many common tasks, it can be considered atomic, e.g., image classification or sentiment analysis of a text should only yield a final label. Models that allow generation of sequences are often auto-regressive, i.e., new predictions are conditioned on previous predictions. Generation for these models is a progressive method and can be subject to steering. For example, language models such as GPT2 [246] allow generation of text phrases, word by word, from left to right. The selection of which word to choose can be subject to human interference. This interaction can influence the right-side progression of building the text phrase. It therefore helps to steer the output of a DNN model in inference mode.

With the recent rise of foundation models, e.g., large language models, interaction paradigms are shifting. The enormous amount of resources needed for training these large models from scratch is paid forward to allow future and cheaper fine-tuning on specific tasks. In addition, a completely new class of task adaptation has emerged: prompting and prompt-tuning, which enable instructing a model to solve a task using only the inference mode. The task instructions can either be given as a prefix in plain text (discrete) or as embedding vectors (soft prompting). Iterations to find the best prompts can be human-driven (prompt engineering) or automated (prompt tuning). The potential of progressive analytics remains to be seen, but one area where it might be found effective is in methods for qualitative evaluation of foundation model fine-tuning and prompt tuning. Human-controlled prompt engineering and time-intensive pre-training from scratch do not seem to be very well suited to the task. An interesting special case of progression that is driven by users can be seen in large language model training. These models are refined by human feedback using a method called reinforcement learning through human feedback (RLHF) [53].

Evaluating the improvement by each RLHF step holds some progressive promise.

Research Agenda: While RLHF methods are natively progressive, it is interesting to investigate how to steer and verify the feedback process.

8.5 Challenges and Recent Advances

Research Agenda: It is possible to report a number of cross-cutting challenges for advancing the progressive machine learning agenda.

- **Algorithm development:** The progressive nature of the computations means that training is done over batches and should be done with increasingly available parts of the data. This requires incremental learning routines that can adapt to changes in the data [184]. The challenge is to determine how effective models can be built using partial data and with minimal storage requirements. Another fundamental challenge for progressive methods is to be adaptive to and robust against dynamically varying phenomena/concepts that change over time (i.e., non-stationarity). Despite being a common challenge for most ML methods, often referred to as *concept drift* [326] or *non-stationary learning* [296], this issue is particularly pronounced in progressive settings due to the progressive manner of the training process.Progressive algorithms should provide plasticity in adapting to changes but they should do so in ways that are not over-sensitive to variation in the data due to the progressive handling of the data.
  Recent developments in online learning techniques [184] explored how some established techniques might be adapted to work in progressive settings [183]. Balancing the stability and flexibility of models in response to different ways that the changes in concepts manifest themselves is another area that has been explored. In addition, progressive solutions such as progressive neural networks [259] offer promising avenues for progressive ML methods. Developed to address problems in transfer learning, these approaches use a model learned in a source domain as an initial starting solution for a new problem domain [62].
- **Model building:** Building progressive machine learning models raises unique challenges during the model-building phase, in particular in relation to parameter selection and the fine-tuning of a model. *Model Architects* and *Trainers* need to make decisions on the design of the model before having access to the full data. For instance, if trial-error procedures are used as experiments to

assess how the model performance varies in response to particular parameter choices, model builders need to make judgments on certain "early" indicators of performance and assess suitability of parameters based on such incomplete and dynamically varying signals. To further complicate matters, progressivity itself introduces its own parameters, such as rate of convergence or number of iterations, which can have implications on how measures of quality and performance are interpreted. **Research Agenda:** Thus, providing informative and consistent indicators of progress and performance assumes the utmost importance, as well as providing mechanisms to alter the progression process interactively during the training process.

Recent developments in the field of active learning, where a human "oracle" provides feedback to an algorithm on data instance level outcomes within unsupervised learning settings [55], offers the potential to enhance the model building process in progressive settings. In such techniques, early indications of model (un)certainty and quality can be utilized to gather targeted feedback from the user (*Architect* or *Trainer*) to guide the model development process during progression. Recent techniques where Generative Adversarial Networks are used to generate synthetic samples to speed up the training process [346], where methods actively learn on data features in addition to data instances [118], and interactive visual frameworks [24], can enrich and strengthen active learning protocols in progressive settings. Interactive visual techniques offer further opportunities to support the model building process by providing approximate solutions that can be steered effectively [233], as well as advanced interactions to explore parameter and feature spaces using progressive visual representations [304]. **Research Agenda:** Such advanced methods of "active learning" can elicit more complex tacit knowledge from people to facilitate more effective steering of models in progressive learning settings.

- **Human factors:** The increasing complexity of most recent machine learning techniques is now known to be a particular challenge for their users due to issues around interpretability, transparency, and trustworthiness [200, 328]. These issues are exacerbated with notions of progression and uncertainty in progressive machine learning settings. Users of progressive ML systems need to assess the status of and outcomes from a model under such uncertain, dynamically changing contexts. This necessitates the careful consideration of human factors in designing progressive ML solutions. It is vital, for instance, to explore how interpretability of ML outcomes [127] can be supported when

"explanations" are dynamically changing and highly uncertain.
Recent techniques in visualization research provide some good examples of how human intervention can be incorporated even in the training of complex ML models such as Deep Neural Networks [127]. Visual representations that show the stability of some inner structures such as perplexity histograms [231] and aggregated views to reveal relations between parameters and outcomes [178] show great potential. Similar techniques are being applied in different application areas, such as health [167] or transportation [341], to advance our knowledge of how users in different contexts respond to such interactive interventions. Methods to allow investigation of foundation models can reveal progressive insights, especially when used for comparing model snapshots [295].
Research Agenda: Further research into how progressive ML methods are used "in the wild" are needed to respond to different needs and diverse stakeholders (see also Section 9.2).

8.6 Summary

With datasets becoming larger and models more complex to design and train, progressive machine learning techniques are likely to be cornerstones in systems where algorithms are in constant interaction with humans, whether it be for learning from humans or for better explaining outcomes. This chapter explored this area of progressive machine learning from the perspective of the humans involved and the characteristics of the algorithms, and presented a number of examples as case studies to illustrate a series of challenges and some promising steps taken in related research areas. Progressivity presents new challenges and exacerbates some existing ones in how people design, train, evaluate, and then use machine learning models. **Research Agenda:** Inherent challenges of progressive techniques involving the online processing of data and rate of convergence become especially important to consider in designing effective algorithms, while issues of quality and uncertainty are key in facilitating functioning interactions between users and the algorithms. We explored several recent exciting areas of research that have begun to tackle these challenges, and expect to see growing activity in all of these and potentially new areas in the development and deployment of progressive machine learning systems. We discussed how recent advances in visualization and interaction research empower various stakeholders in designing, training, and using early instances of progressive machine learning techniques, and we expect visualization to continue playing a key role as an enabler for the development and adoption of progressive systems.

9 Evaluation

9.1 Scenario and Challenges 150
9.1.1 Approaches to Evaluation 150
9.1.2 Performance Measures 152
9.2 Evaluation at Multiple Stages of the Pipeline 153
9.2.1 Data Management 153
9.2.2 Algorithms 156
9.2.3 Visualization and Interaction 158
9.2.4 Human Factors: Perception and Cognition 163
9.2.5 Machine Learning 166
9.2.6 System Level 167
9.3 Summary 169

Author Gaëlle Richer, Jean-Daniel Fekete, and Michael Sedlmair

Designing progressive data analysis systems affords several degrees of freedom. Algorithms, systems, and techniques require evidence to demonstrate their feasibility, effectiveness, and usability. While this characteristic holds true for all visual analytics systems, the dynamic nature of progressive systems poses new challenges for the creation of this evidence due to the tighter connections between the two classes of systems involved: visualization and progressive data processing. The approaches to evaluating visualization and progressive data processing systems are fundamentally different and often incompatible. Evaluating visualization systems usually borrows methodologies from Human-Computer Interaction (HCI) and psychology, while progressive data processing utilizes standardized performance benchmarks. **Research Agenda:** Investigating the performance of each class of system individually is near-meaningless; instead, an integrated evaluation must be performed to demonstrate the value of a progressive data analysis system, similar to what is customary in evaluating visual analytics systems [235].

At the highest level, progressive systems offer a natural trade-off: wait time versus quality and confidence in the results. At one extreme is simply waiting for the results, and the task of helping humans with the waiting process is related to wait-time engineering [80, 278] (also discussed in Section 7.1). At the other extreme is the temptation to make decisions too quickly, regardless of the quality of the results. This time/quality trade-off can be critical in time-constrained exploration [161] and also important in other contexts, since time is a precious resource that should be used wisely. We want to build interfaces where users can make accurate decisions as soon as they have meaningful results that they can have confidence in—but no sooner.

At a lower level, progressive systems should efficiently manage both human and machine capabilities. Humans have limitations in their attention span: they have great difficulty remaining engaged with long latency, and too many blinking visualizations that have to be monitored can overwhelm their perceptual and cognitive capacities. Therefore, progressive systems should strive to ensure that human attention remains effective. On the other hand, some algorithms are effective but return results that do not comply with human capabilities, such as unstable intermediary results that are hard to track visually. Evaluating solutions that trade speed/quality and are compatible with human capabilities adds further levels of complexity.

9.1 Scenario and Challenges

Assuming that progressive systems become commonplace in the visualization & analytics system landscape, we can imagine that a typical evaluation scenario might be to demonstrate the effectiveness of a system, algorithm, or interface to (1) allow something that was previously impossible or (2) improve an existing progressive solution. For the former, case studies can describe the situations and problems that can be tackled now and why they could not be tackled before (e.g., [23]). For the latter, the important point for evaluation is to define the dimension(s) of success under consideration: the ability to make early and confident decisions, the ability to make decisions in a time-constrained situation, to save compute power (green computing), etc.

9.1.1 Approaches to Evaluation

Some papers evaluate the utility of a novel progressive technique (with a case study or observational study) or compare its performance against an eager system, i.e., a non-progressive equivalent that performs a complete computation before returning results (with a user-controlled study or algorithm performance comparison). In most cases, existing work targets both the time and the accuracy of user decisions.

In the best case, a progressive algorithm will be the fastest, as in time-series search from Gogolou *et al.* [102] and Progressive Topological Data Analysis with Vidal *et al.* [316]. Will evaluating the speed then be sufficient? If the algorithm is used in a non-progressive context, the speed is sufficient, but in a progressive context it should also be evaluated according to the requirements mentioned in Table 2.1. These requirements are important, but they also introduce concepts that are not well specified, such as: *meaningful* partial results (R1), *structure-preserving* intermediate results (R2), *retaining cognitive workflow* on update (R3), *minimized distraction* during updates (R4), and *cues* for new results (R5). **Research Agenda:** Assessing whether a progressive system meets these requirements will require their authors, and ultimately the PDA community, to clarify these concepts and state that their system implements them fully, partially, or given specific definitions. To support the quantitative evaluation of requirements fulfillment (such as for R1 and R2), there is a need for dedicated quality measures in alignment with human judgments [274].

In general, **Research Agenda:** a first step when describing evaluation goals would be to clarify the contextual properties of the analysis: the data, the human context, and user involvement being considered, and whether the main task is one result (with a sequence of approximations) or of a search/gather nature which outputs an extending list of results, e.g., for exploring frequent sequences of temporal patterns [293].

For instance, regarding the expected stability of intermediate results, to clarify the contextual properties of the data involves assumptions about the bounds of the data (known, unbounded), the data stationarity nature (shuffled or not), and whether the user can steer the computation. In terms of the human context, we need to explain what we expect to be a satisfying result: the level of desired uncertainty and the contextual trade-offs between time and quality. Clarifying the tasks and data is necessary, since a progressive visualization or computation could be such that its intermediate results do not support a task, while its final result does. For example, the progressive visualization of sorted bar charts can be very unstable when bar values are nearly identical, yet may provide a usable final result once the values converge. The finding of an evaluation may greatly depend on the decision-making strategies that are adopted or communicated to users, including the goal trade-off between time and accuracy.

Many existing methodological approaches, such as measuring runtimes, running controlled experiments, and conducting design and field studies, will remain relevant for PDA and PVA. At the same time, new methodological challenges arise from the need to evaluate progressive tasks, such as (1) monitoring the unfolding of a complex process to understand its inner workings and make early decisions (e.g., detection of errors, unsuitability of the process, clear-cut result), (2) refining the scope of an analysis question or steering the computation based on an approximate overview, or (3) conducting branching or speculative analysis based on intermediate results. **Research Agenda:** These tasks are challenging to evaluate because they involve a trade-off between wait time and confidence in the results, early-stopping [52], and steering of the progressive process.

9.1.2 Performance Measures

Performance measures that evaluate the ability to make early and confident decisions apply to the computation and user levels. The measures of performance for a progressive computation task include:

- bounds on time from start to first *meaningful* results
- time until first *accurate* results
- time until the *final* results
- *stability* or *convergence* of intermediate results;
- frequency of substantial result updates, and interface responsiveness
- relevance of the uncertainty/quality metric.

In the case of a progressive search/gather task (defined in item 2.1), intermediate results do not improve but expand, and measures of accuracy translate to measures of specificity and relevance. From a user's point of view, measures of performance also include:

- behavioral and psychological factors (e.g., maintaining attention and limiting fatigue)
- assessment of uncertainty, stability, and progress
- latency, amount, and accuracy of derived insights
- subjective level of confidence in results.

Progressive systems should also be evaluated, when appropriate, against human bias, as well as data and machine artifacts linked to progressive visual analytics, which can cause decision errors. One example we discuss in Section 7.5 is the illusion bias, which describes the lingering effect of a false intermediate pattern on the user, such as a clustering pattern [197]. One example of data artifacts can be the non-stationary dynamic of a sequence of chunks: if the chunks are not properly shuffled, the user might be lulled into a false sense of stable results and derive erroneous conclusions. Similarly, early results may look more certain than they truly are: a low-quality clustering result that depicts an overly prominent grouping of the data might mislead a user into thinking a cluster will be more important than it really is.

Research Agenda: The definition of user performance measures beyond time, accuracy, and confidence is one of the challenges of evaluating progressive systems. Additionally, user studies are needed to understand the landscape of guarantees from the underlying progressive processes and how they impact user perception, cognition, and decision. For example, what guarantees should be expected regarding latency and error bounds for intermediate results? These questions also relate to how uncertainty (R7) and progress (R9) are visually conveyed.

9.2 Evaluation at Multiple Stages of the Pipeline

Evaluation questions and methods take different forms along the visualization and analytics pipeline. In the following, we provide examples of measures of success and evaluation challenges for building blocks of the progressive pipeline.

9.2.1 Data Management

Evaluation of data management systems is concerned with the efficient management and distribution of more or less structured files. The criteria listed in Section 9.1 are important in serving queries. But the data management layer is also concerned with preparing the data to improve progressive computations, including indexing, but also more specific types of capabilities related to progressive computation, such as *shuffling* and *properties estimation* as mentioned in Section 3.6.

Properties Estimation

When data is queried progressively, data aggregation functions that are used to return point estimates in standard SQL (e.g., mean) can return confidence intervals or probability distributions to express the uncertainty of the running results. The evaluation of progressive queries should be clear about the expected return values computed since maintaining confidence intervals using, e.g., bootstrapping [76] or maintaining a value distribution are expensive operations that impact the performance of the queries and the complexity of the data structure returned.

It could also provide an estimate of the data distribution and various levels of statistics related to the queried data. Providing these can dramatically help the algorithms downstream, and therefore, progressive data management can be evaluated according to the richness of the properties that systems are able to provide to improve their accuracy while answering queries. For example, a web-based data management system can return a streaming CSV file, sometimes applying a simple SELECT filter on the fly. A PDA application would need to progressively compute the min/max column values over the stream. A visualization of that stream would be unstable since the data scale would change each time a new minimum and maximum value is read from the stream. Such a system would be usable for PDA but would be unstable, and data would arrive in an unpredictable order, probably not shuffled and likely with a heavy trend, slowing down any downstream algorithm. A better system would provide additional capabilities serving the visualization, such as shuffling to avoid trending. But it would also send the min/max values upfront if possible, or, it would do so in a progressive way, faster than the progressive computation on the client would, allowing the visualization to be stabilized faster. An even better system would send distribution or quantile information in addition to min/max to further stabilize the visualization. Most big data tables have outliers that limit the utility of the min/max values and should use a scale based on the [5%,95%] quantiles. We expect the user feedback to be different for these varying levels of service of the PDA system, from a raw to a smarter and richer data streaming strategy. Ultimately, an evaluation should report on the effect of these differences.

Data Management Evaluations

Battle at al. [19] discuss the problem of evaluating visual data analysis systems from the database and visualization community viewpoints. They mention three levels of evaluation: component, system, and work environment, and suggest a vision for new evaluation methods based on benchmarks and a trace repository.

As a follow-up and implementation of the vision for systems, Battle *et al.* [21] introduce a benchmark for evaluating interactive visualization applications served by a database. Their benchmark addresses continuous dynamic queries, requiring a latency below 0.1 s, with latency being the time between issuing a request and receiving its response. Their evaluation criteria are *responsiveness* and *accuracy*. Responsiveness assesses whether the returned results are fast enough and whether there are latency violations. They do this using two derived measures: *Response Rate* relative to the latency requirement, and *Average Query Duration*. Accuracy is assessed because, in progressive or AQP systems, “approximate results deviate from the ground truth, introducing error into the resulting visualizations and possibly misleading users” [21]. Accuracy is measured by the *Mean Relative Error*, which is the ratio of the error to the expected true result within a given time period. Finally, performance is also measured by *throughput*, the number of transactions processed per second. For large datasets, their benchmark shows poor performance with traditional non-progressive database engines, whereas with the VerdictDB progressive database [223] the system remains responsive and the accuracy is always acceptable.

Eichmann *et al.* [77] introduce a benchmark for interactive data querying systems—including progressive ones. The paper explores a number of factors for the time to interact with a dataset, including *data preparation time*: the time to load data into the database engine before it can process queries. They separately mention the support of *table joins*, since some systems use it heavily and others do not support it (see Section 3.4). The paper describes a metric composed of 7 measures: *time requirement violated*, *missing bins*, *mean relative error*, *cosine distance*, *mean margin or error*, *out of margin*, and *bias*.

Supporting joins is challenging, as noted in Section 3.4. Procopio *et al.* [242] discuss the specific problem of progressive joins, implemented using the "Wander Join" algorithm [243]. It is evaluated using the standard TPC-H [300] database benchmark in terms of relative error over sample size on a typical query. This evaluation highlights the difficulty of comparing absolute time in progressive algorithm benchmarks when comparing experimental systems with optimized implementations of non-progressive systems, hence the use of "sample size" instead of time. The behavior of their progressive algorithm is promising, but suffers from the fact that the TPC-H benchmark provides data that is not representative of exploratory data analysis workloads, and their implementation is not optimized.

To summarize, the main evaluation criteria mentioned in the articles are: *throughput*, *latency*, and *accuracy*, with several measures for each of them. In most cases, the evaluation is done using prepared datasets, i.e., indexed and prepared for fast shuffling. Since data management research relies heavily on benchmarks rather than ad-hoc datasets, the evaluations are performed on traditional TPC-H benchmarks or newly gathered benchmarks based on user traces. Unfortunately, both of these have flaws: TPC-H is not designed to be representative of an exploration workload; and the new benchmarks are not yet standardized (nor particularly diverse in their use patterns). **Research Agenda:** It would be very useful if more diverse benchmarks were developed with both representative data distributions and queries.

It is less clear how to evaluate the additional capabilities of data management systems for progressive applications, which can greatly influence the performance of analytical pipelines, as described in Section 3.7: in-situ systems allow processing of raw data immediately without a long ingestion process, as well as bi-level shuffling [50].

Other aspects of progressivity in data management have yet to be evaluated from a data management perspective, such as *query steering*. However, query steering has been evaluated from a user perspective. Selective Wander Join [242] implement query steering through importance sampling, driven by weights on the different groups in the query that can be interactively adjusted. They evaluate this steering capability at the system level, in an expert user study, rather than in the broader context of a progressive database system.

Research Agenda: More work will be needed to devise new evaluation methods for several aspects of progressive data management, in particular regarding human factors.

9.2.2 Algorithms

Evaluation of algorithms in the analytical part of the pipeline, or even in the visualization part for network layout or multidimensional projections, is concerned with (cumulative) resource usage and time to the final result. Like online algorithms, progressive algorithms can be evaluated with *competitive analysis* [30], comparing their time and memory consumption to their eager version [48]. They are also evaluated with other criteria specific to the progressive setting:

1. time to early meaningful results
2. latency between the intermediate results [142]
3. accuracy and precision or uncertainty/quality of the intermediate results [206, 304] for algorithms computing one result, or
4. recall and precision for algorithms searching for many results.

As introduced in Section 2.3, progressive algorithms can be split into two groups: *progressive-in-chunks* algorithms that return one result computed progressively—i.e., a result is not only a literal, but can also be compound values such as a vector, a matrix, or a table—and *progressive-in-iterations* algorithms that search/gather multiple values progressively. We discuss these here separately.

Computing One Result In eager algorithms, the main evaluation measures relate to accuracy (how close a given set of measurements are to their true value), precision (how close the measurements are to each other), computing time, and consumed resources. Progressive algorithms that compute one value can be sensitive to data stationarity. Therefore, papers on progressive algorithms need to specify the assumptions they are making or take counter-measures to continue working in degraded environments. Progressive algorithm evaluation borrows methods from online algorithms and streaming algorithms. Online algorithms are usually evaluated using *competitive analysis* [30]. Streaming algorithms are often evaluated according to their low memory requirements, in addition to their accuracy and speed [212]. While competitive analysis is informative for progressive algorithms, it is not sufficient; other properties are also critical, such as the real-time constraints and steering capabilities. As for streaming algorithms, they are evaluated with strong assumptions of data stationarity, which are not always simple to enforce for progressive computations. Additionally, progressive algorithms should assess the quality/uncertainty of their results. **Research Agenda:** Therefore, there are opportunities for continuing to develop methods to articulate assumptions and evaluate progressive algorithms.

There are several examples of the evaluation of progressive algorithms that compute one result. An interesting example is Xin *et al.* [48], who evaluate progressive 2D sampling for scatterplots with several methods. These use competitive analysis, comparing the quality and time between the static version and the progressive version. They also compare running the static version frame by frame to the progressive version, trying to be as stable as possible between frames. The results show that the progressive version is both faster and more stable, but accumulates errors to maintain stability. This error can be shown to the user, who can decide to interactively recompute the sampling to optimize it rather than keeping it stable (requirements R2 and R13). A few other algorithms [142, 293] consider interactive interventions to allow the user to choose between accuracy and stability. **Research Agenda:** It is not clear how interactive interventions should be managed in progressive algorithm evaluation.

Overall, evaluations of algorithms focus on time and accuracy, but they start from different assumptions (data fully loaded, kNN fully computed, stationary data, static vs. dynamically filtered data). The results are presented with absolute time or number of iterations, and an iteration's meaning should be formally specified to be reproducible. Finally, several algorithms provide interactive methods to allow the user to trade stability for quality; such user interventions make it more complex to evaluate the algorithm.

Searching/Gathering Multiple Values A progressive search/gather algorithm returns a list of results that grow or improve with time (item 2.1). For a simple database query, more results will come with time. In eager algorithms, the main evaluation measures are related to precision (the fraction of relevant instances among the retrieved instances), recall (the fraction of relevant instances that were retrieved), computing time, and consumed resources. In PDA, the quality is also related to completeness and to the probability of finding more relevant results by waiting for more. In data mining, searched/gathered results may be more or less interesting (frequent or rare), and completeness relates to more patterns as well as more interesting patterns.

There are a number of early studies of progressive systems that gather multiple values. Stolper *et al.* [293] propose a system, “Progressive Insights,” that is designed to search for common patterns in event sequence data. The system progressively returns a growing set of matches. Their evaluation is qualitative: they report use cases and observations of a doctor using the system and commenting (positively) on its capabilities. Ravenau *et al.* [248] also report on sequential pattern mining, describing six tasks they want their system to support. They evaluate their system using competitive analysis [30], showing that their progressive version is slightly slower but more flexible than its eager equivalent. They also performed a controlled study with 14 participants to compare their interface with and without steering. They show that participants with steering were both faster for their tasks and were also more accurate with their answers. They also report on feedback after the controlled study using a questionnaire and show that users preferred a progressive system and its steering capabilities. Gogolou *et al.* [102] report about a method to perform data series progressive similarity search. They evaluate their method using synthetic and real datasets and evaluate the quality of results as well as the time savings of the early stopping strategy. For the quality of results, they measure how much of the total search space has been covered over time and evaluate the confidence intervals used in finding the best match. Another aspect of the quality of results they evaluate is the time guarantees for the exact answers. The time savings are measured by the time saved using early stopping relative to the exactness of answers. Their evaluation does not take into account the benefit of progressive intermediate results returned, only the quality of the final “best” result, which remains the recognized criteria in the database literature.

All the articles on search/gather techniques mention useful properties, such as early partial results, but they do not evaluate their importance specifically. **Research Agenda:** There are substantial opportunities to refine methods and methodologies to evaluate search/gather progressive results. A promising theoretical framework for this is *optimal stopping* [52], which consists of choosing a time to take a particular action in order to maximize an expected reward or minimize an expected cost. However, choosing a good reward/cost function might not be obvious for exploratory tasks.

9.2.3 Visualization and Interaction

Evaluation of progressive visualization and interactions is concerned with enabling user interaction during progression (R12), and it seeks to provide effective visual feedback on computation progress (R5, R11) to support users in generating informed insights and making early observations/decisions. In some cases, the eventual completion time of the progressive process, i.e., the time that is waited for the final results (R10), is also an important factor. To support users in decision-making, other properties of the progressive visualization can also play a role, including how fast the progressive computation converges and how users visually assess the quality of the visualization at a given step (R7).

We can learn about the evaluation of progressive visualization by looking at analogous systems in computer graphics more generally. Progressive rendering has been investigated for use in tools for editing 3D realistic graphics, such as the shapes of computer-aided design (CAD) models or lightning and materials of a movie sequence for computer-animated films. Interactive previews are essential for editing tasks that require making careful enhancements while being mindful of images that take hours to render. For example, progressive rendering is increasingly becoming a standard in relighting engines [267], systems that allow users to interactively configure and refine lights in a scene. The primary challenge is to provide adequate fidelity in the initial approximation to enable an initial placement and configuration of lights. Although many rendering algorithms enable straightforward, progressive updates, methods differ in how predictable they are and how fast they converge for complex scenes [267]. These tools are evaluated following an established user methodology where users run both open- and non-open-ended tasks in the interfaces being compared. Measures of success are both objective (time to completion, image error) and subjective (user ratings for the effectiveness of specific tasks and globally) [267]. Some work has also focused on improving the refinement strategy of rendering algorithms to match the vision model [81], suggesting perceptible image error as an alternative success criterion. **Research Agenda:** Similar user methodologies could be developed for comparing progressive visualization methods in the future.

Progressive visualization introduces dedicated visualizations and interactions designed to help users make sense of the progression, assess intermediate results, and manipulate intermediate views. Existing systems often convey feedback on progress and quality using separate visual views, such as progress bars for progress and line plots for the quality history (e.g., [16]). Evaluations focus on the utility and effectiveness of a new visualization, in some cases in comparison with an alternative eager interface. Measures of success include usability scores, insights [342], and interaction metrics, time and accuracy of user answers, user confidence, and criteria for visual stability [224]. Patil *et al.* [224] (also discussed in Section 6.3) perform a controlled user study for progressive bar charts using multiple conditions and also compare human results with those of sequential statistical tests (humans are good) that remain heuristics for data with unknown distributions.

Usability Progression and quality feedback (R10, R11) are essential to users in assessing the trustworthiness of the visualization, and support them in making decisions with confidence and at an appropriate time. In particular, any change in the context of application and tasks, e.g., toward more sensemaking or more time-pressure [161], will influence how a user assesses the time-quality trade-off and the time point at which they determine that they can or need to decide. Subjective feedback collected during user studies showed that simple progress bars were found to be useful to users overall [16]. Progress feedback can take different forms, and thus different visual representations, depending on the type and complexity of the upstream progression process (relative or absolute, estimated or not). Therefore, checking to make sure that users can make sense of the progression is an important aspect of the evaluation.

There can be multiple progress indicators calling for new representations, for instance, when users have the ability to start multiple processes in parallel. In ProReveal [141], users can compose multiple progressive charts, although only one can run at a time. A side-view list summarizes the progress of every ongoing visualization by displaying progress indicators, and allows alternate computation between visualizations. The major focus of the interface is interactive guards, called PVA-Guards, which are operators and visual reminders whose purpose is to help conduct speculative analysis of the data. Users can formulate their current hypotheses and register them to the system such that their validity is tested continuously as intermediate results improve. The guards are visually displayed on the side and provide a notification if the hypothesis proves incorrect. As a proof-of-concept, PVA-Guards have been evaluated in a user study using usability as a measure of success: testing whether users were able to correctly interact and gain insights, while collecting subjective usability ratings. Finally, usability studies also report user comments, observations, and descriptions of their preference [16, 304, 342].

Research Agenda: With regard to usability, there is also a need to evaluate how well system or algorithm guarantees, such as probabilistic bounds on the error in intermediate results, are visualized, understood, and leveraged by users in their analysis strategies.

Time, Accuracy, and Confidence Together with indicators of progression, indicators of uncertainty are essential components in making informed decisions (R7). As presented in Section 6.5, the existing body of work and guidelines on uncertainty visualization can be utilized in progressive visualization, such as traditional static error bars or transparency coloring. Patil *et al.* [224] compare four uncertainty representations for progressive bar charts, showing that the representations using error bars and near-history are significantly superior to the baseline bar chart and the combination of both error bars and near-history, for simple tasks. **Research Agenda:** More work is needed to evaluate how well these visual indicators are understood in the progressive context, since they are critical for making informed decisions.

Generally, there is a tension between visual complexity and the display of local or fine measures of uncertainty (e.g., single quality measure vs. per-element measure). More advanced techniques use interaction to provide answers to specific queries with corresponding uncertainty. The interactive annotations by Ferreira *et al.* [85], for instance, support various comparative tasks on bar charts. To compare each bar to a range of values, for example, the system estimates the probability that any given bar is inside the user-selected range and displays it using a color scale. These interactive annotations have been evaluated for answer accuracy and user confidence, and have been shown to improve user confidence compared to confidence intervals.

In a set of controlled experiments, Procopio *et al.* measured the effect of properties of progressive results (e.g., false patterns or high variations) and user steering ability on accuracy, confidence, and completion time for progressive gradient plots [240]. Their experiments examine the four types of cognitive biases associated with progressive visualization (see Section 7.5) and are an example of a precise experiment design.

In evaluations using time, accuracy, and user confidence as performance measures, we find a resemblance with the measures commonly used for user experiments of non-progressive visualizations or interfaces. However, time and accuracy have, to some extent, a different meaning in PVA: when completing a task in a progressive system, time is not only used to complete the task but also to wait for the result quality to improve. To evaluate how well users are able to balance waiting time with accuracy, user performance should be compared to those of optimal or near-optimal strategies, such as a sequential statistical test appropriate for the given task [321].

In addition to user confidence, it is also important to evaluate the level of confidence inspired in users with respect to the quality of results, i.e., build and calibrate trust [109]. Evaluating this relationship provides information on how efficiently progress and quality are being conveyed, but it also greatly depends on the literacy of users in the statistical tools on which uncertainty indicators are built. Evaluations of uncertainty visualization, surveyed by Hullman *et al.* [131], follow two main approaches: measuring user accuracy and collecting user feedback on how helpful the visualization was in making decisions. Authors of the survey provide recommendations drawn from other fields that should also be considered when designing human experiments for progressive systems, such as using decision frameworks to create a realistic decision context for participants. Some examples of user measures from the decision-making literature, beyond time and accuracy, are decision consistency and utility maximization [131].

Research Agenda: More study is needed to determine the trade-offs between different measures of user performance in the particular context of user experiments for progressive visualization as compared to current evaluation practices.

Visual Stability For progressive visualization that computes a layout, visual instability can pose problems for usability (R2, R13, see also Section 5.2.1). Preserving the general structure of the representation between updates is important so as not to interfere with users' cognitive flow (R3) and to help them spot changes. However, there is generally a tension between accuracy and stability. To support visual stability, new results can be introduced through animated transitions and changes with color or other cues for emphasis.

Visual stability is often part of the design goals (e.g., constraining the spatial area of change [233]), but it can also be needed indirectly to assess the convergence of the visualization [16, 230]. For layout and clustering algorithms, a quantitative measure related to visual stability is usually computed (e.g., looking at the MDS criterion value over iterations [51]). Giving users control over updates, pause and rewind can compensate for a lack of visual stability [16]. Angelini and Santucci [10] propose multiple data and visual metrics to characterize stability. They propose a preliminary framework comparing visual stability and visual consistency, where consistency represents the secondary goal of preserving the coherence of the visual and data changes between updates.

Currently, progressive visualization evaluations are, for the most part, conducted at the system level, evaluating progressive visualizations and interfaces in the context of a particular processing system. **Research Agenda:** In order to evaluate visualizations in controlled studies, a characterization of the outputs of progressive algorithms and progressive database queries is needed so that the efficiency of progressive visualizations and interactions can be compared more precisely and generalized to a variety of contexts.

9.2.4 Human Factors: Perception and Cognition

Ultimately, progressive analytics systems are meant to be used by people. As such, an evaluation of human factors as described in Chapter 7 will play an essential role. Humans bring perceptual and cognitive skills that can be leveraged throughout the analysis process, but they also have their limitations and biases that need to be taken into account (see Section 7.5). Compared to traditional systems, the incremental changes in a progressive visualization may give rise to additional errors related to perception and cognitive factors. At the same time, perceptual studies will help us to provide guidelines for setting the many open design parameters in a progressive visualization system.

Methodologically, these evaluations will need to rely on quantitative and qualitative human-subject studies, as commonly used in the area of human-computer interaction (HCI) [169] and cognitive psychology [159]. While the primary focus of studies in HCI is on usability, studies that more specifically target perceptual and cognitive models will likely also need to leverage research methods from these areas. Elliot *et al.*'s design space of vision science methods for visualization research [78], for instance, provides a good starting point for the use of vision science methods in visualization and visual analytics research.

Changes over Time A core characteristic of progressive systems is that (intermediate) results are gradually changing over time. In streaming data scenarios, we can simply show new data as it comes in. But in progressive scenarios, it is already clear that this endeavor poses many challenges. From a human factors perspective, for instance, we need to ensure that users are aware of these updates (or are deliberately not being made aware). Furthermore, intermediate results not only introduce changes in results, but also in their associated error bounds or uncertainty measures, and they do not always follow the expected trend.

Change blindness [286] refers to a well-known phenomenon of human perception: under certain circumstances, people will fail to spot a change in a stimulus despite it being obvious as soon as it is pointed out to them. But even aside from change blindness, changes might simply not be visible to a user [27]. For instance, the magnitude of the change in the data might not be commensurately reflected in the visualization [158], and as such, a change might be missed. Likewise, a slow progression of small changes may simply go unnoticed. Studies on Just Noticeable Differences and Weber's law, such as those of Harrison *et al.* [110], are relevant in this context. **Research Agenda:** Human subject studies are needed to investigate these effects in progressive visualization and to systematically consider different ways to counteract them.

These considerations also call for explicit investigations into the question of how frequently changes should be shown to the user. This frequency might be too slow or too fast [38]. Too slow will lead to an interrupted workflow [41] or missed changes. Too fast might overwhelm the user or, counter-intuitively, even slow down decisions [38]. While in the election example of Section 1.1, we would likely show new data as soon as it comes in (possibly too slow), in other scenarios we might have new results coming in on a millisecond basis. In this case, we would need to evaluate different strategies and frequencies in deciding how frequently to update users in progressive systems. These studies will not only need to take into account user performance in analytical tasks but also health and fatigue-related issues [141, 278]. Outcomes where users are either exposed to many hours of rapidly-flicking updates, or stare at a static screen showing no changes at all, are equally to be avoided. Even in conventional eager contexts, the effects of latency are not yet well understood [180], with some results showing that, while affecting user strategies, they are not always detrimental to user performance in higher complexity tasks [20]. In the progressive context, systems might offer updates automatically, or, following "anytime" strategies (Section 2.3), could allow users to request changes on demand, enabling even more complex user strategies. However, it would be ensured that the user is not missing important updates, which calls for studies on peripheral vision in the context of progressive systems.

Mental Models and Sensemaking A central aspect of traditional visualization and visual analytics systems is to support sensemaking [234], that is, the process of lending meaning to the data that users investigate (see also Section 7.2.2). Sensemaking will also be a key task for progressive analyses. However, since new results can come in progressively, the classical sensemaking process might undergo substantial changes in such setups [342]. Instead of operating on a static problem, users might engage in speculative exploration, more rapidly coming up with new hypotheses about the data, but also needing to correct or "up-believe" them [100, 101]. In the process of sensemaking, mental models (see also Section 7.4) play a key role, that is, the users' internal explanation of how something works. Rapid changes could be problematic in the slower process of coming up with coherent mental models about the data. The faithful perception of uncertainty will also be key in this process. Further studies will be needed to understand these exciting new facets of the sensemaking process. **Research Agenda:** While studying visual sensemaking is already a non-trivial endeavor for traditional visual analytics systems [151], progressivity will present new challenges.

Risks and Biases As described in Section 7.5, properly understanding and potentially mitigating cognitive biases will play an important role in progressive analytics. There have been many studies on such biases in general [149], but also some specifically addressing visualization and visual analytics [68, 310]. There is a need for more studies that specifically consider the characteristics and specificities of progressive systems, following the experiment on cognitive biases of Procopio *et al.* [240]. **Research Agenda:** How exactly do these biases manifest in progressive setups, how severe are they, and how effective might different countermeasures be? Are the initial results anchoring or priming the user [309]? Is the user subject to illusion biases in the presence of false patterns? These questions necessitate a careful and rigorous empirical investigation. While initial hypotheses can be derived from existing theory as described in Section 7.5, empirical studies will need to confirm, refine, or refute them.

Decision-making Evaluations of progressive systems also share similarities with decision-making environments. The decision-making literature uses consistency of decisions as a success measure [32], which is also relevant to progressive systems. Measures of user performance from progressive system experiments are most likely strongly dependent on decision strategies followed by participants and what they consider to be good enough responses. As a consequence, evaluations need to report on strategies deployed by users to mitigate any waiting time between updates, as mentioned previously, but should also clarify elements that affect how participants calibrate their goal in terms of the trade-off between accuracy and response time. This concerns any scenario or domain requirements that create specific time pressure or implicitly define acceptable thresholds, but also the reward mechanisms to financially incentivize participants. An example of a decision-making strategy is "satisficing", where the user first defines acceptable thresholds in advance and then chooses, or stops at, the first alternative that satisfies the thresholds [67]. Decision strategies applying to progressive tasks differ from those described in non-progressive contexts, since progressive tasks can include strategic waiting time to increase accuracy. **Research Agenda:** Methodologies are needed to describe decision-making requirements and strategies, and assess the optimality of user behaviors. Part of a system's evaluation would also be to determine how well it performs under different requirements (e.g., fast decision or minimum accuracy).

9.2.5 Machine Learning

As discussed in Chapter 8, progressive machine learning can be used by Architects, Trainers, and Model Users. The evaluation criteria for the three groups are different. For the Model User using a pre-trained machine learning algorithm, evaluation is very similar to progressive algorithm evaluation. For the Architect and Trainer, few progressive machine learning algorithms are reported in the literature, although most of the iterative algorithms could be made progressive, and online machine learning is also a first step toward progressive machine learning. Machine learning algorithms are diverse; a typical machine learning pipeline is made up of several stages, e.g., data preprocessing, feature preprocessing, and an estimator [87]. Data processing stages belong to standard algorithms, whereas we consider feature processing and estimators as machine learning algorithms.

Progressive machine learning has been evaluated at the algorithm and system levels. With the advent of explainable AI, **Research Agenda:** they should also be evaluated regarding their capability of explaining or adding transparency to the learning process. Several machine learning algorithms have been evaluated at the algorithm level. Many of them are dimensionality reduction methods that are feature processing methods for machine learning, but can be used for visualization in an exploratory context. Several articles on progressive algorithms highlight their capability to return a progressive result at an acceptable pace [94, 233, 334].

PCA is one of the most popular dimensionality reduction methods. Fujiwara *et al.* [94] describe a streaming PCA algorithm that can be considered *progressive-in-chunks* and tries to "maintain the user's mental map". Stability is critical with the Incremental PCA algorithm [255] because the sign of the eigenvectors can flip at each iteration, resulting in a strong visual disruption if not controlled properly. Fujiwara *et al.*'s technique is useful for computing uncertainty and stabilizing the visualization. Their algorithm is evaluated using the run-time performance of the algorithm and two use cases. Streaming PCA is not progressive by iteration natively since it does not guarantee a quantum, i.e., a time limit for its execution, but by tuning the size of its chunks its latency can be controlled.

Pezzotti *et al.*Approximate t-SNE (At-SNE) [233] is evaluated based on precision over time, the number of points handled over time, and a specific quality measure of t-SNE over the number of iterations. This latter measure facilitates reproducibility and replicability of the article's evaluations, as running times are not reproducible. However, At-SNE is only *progressive-in-iterations* after the approximate k-Nearest Neighbors (kNN) of the dataset has been fully computed. In contrast, Jo *et al.* [142]

describe a *progressive-in-chunks* kNN algorithm and implement a *progressive-in-chunks* and *progressive-in-iterations* t-SNE algorithm called Responsive t-SNE as a use case of their progressive kNN. As intended, this implementation is fully progressive and allows for monitoring the computation of a multidimensional projection when data is loaded progressively (*progressive-in-chunks*). The evaluation reports absolute loss over iteration instead of relative loss over time. Each of the two versions of progressive t-SNE implements specific features, such as computational steering for At-SNE and interactive exaggeration for Responsive t-SNE. Steering allows users to devote more resources to faithfully compute the projection around a point of interest. Interactive exaggeration applies a global optimization phase at the user's initiative, since it could disrupt the visual stability of the progressive projection if triggered automatically (requirements R2 and R13). These features are described in the articles but not evaluated in any way. Finally, Responsive t-SNE is designed to recover from non-stationary input. Its progressive kNN algorithm adapts to the non-stationarity and influences Responsive t-SNE to fix its computations when needed. This feature is not formally evaluated, but images of the progression are shown as a visual assessment of the process (see Figure 4.2). Showing images of the results is an accepted assessment in computer graphics and visualization, although newer articles tend to prefer computational evaluation approaches. However, judging the interpretability of a set of progressive results is currently ill-defined and can only be done with human judgment.

Several progressive estimators are mentioned in the literature. Jo *et al.* [142] show examples of kNN regression and density estimation and can easily be used to implement a kNN classifier, but none of them are evaluated. Fekete and Primet [83] mention a k-means clustering module in their ProgressiVis toolkit but also do not evaluate it.

9.2.6 System Level

Due to the lack of characterization in the requirements for the data processing, machine learning, and visualization parts of the pipeline, many PDA approaches are evaluated at a system level. Since conducting controlled studies for entire systems is difficult, a common evaluation approach is to conduct case [283] or design studies [277] with experts. These evaluations are typically conducted in an open-ended manner and report on observations made during interviews with domain experts, while or after they use the system. For example, for the sampleAction system [89], experts reconstructed a question they had recently addressed, or could have addressed, and were asked to give their interpretation of intermediate results at several points of the progression.

A specific form of this type of evaluation is Multi-dimensional In-depth Long-term Case studies (MILCs), proposed by Shneiderman and Plaisant [283]. For instance, Stolper *et al.* [293] conducted a two-month case study with a domain expert following the MILC protocol of Perer and Shneiderman [227]. This protocol has five phases: an initial interview, training, a first usage phase with possible modification of the software, a second usage phase, and an exit interview in which the expert can report on their level of achievement for each goal originally set. During the usage phases, observations are collected. This approach provides a longitudinal and in-depth understanding of how experts use the tool and provides guidance on the redesign of the tool. For DimXplorer [304], observations were collected for four analysis sessions over two months. During interviews with experts, authors seek to interpret progressive results and generate preliminary insights.

Design and evaluation aspects are commonly combined for these types of projects, resulting in design study contributions [277]. Systems are iteratively evaluated along the entire design process in close collaboration with domain experts. The qualitative reflection on this process allows then to improve, extend, or challenge existing design guidelines. Design studies are very common in the visualization literature, and examples range from reports on systems designed for biologists [195], system managers [194], and automotive engineers [276].

Other systems are evaluated based on controlled user studies of non-domain expert users working on both open- and non-open-ended tasks, for instance, using Saraiya *et al.*'s insight-based methodology [265]. For InsightsFeed [16], the experiments collect quantitative measures, subjective ratings, and qualitative observations collected from user recordings. Measures of success include the ability to make early observations, to answer with confidence, serendipitous observations, and usability rate for different UI parts. Zgraggen *et al.* use a hybrid experiment design with both log-based and speech-based metrics to measure insight and interaction rates, insight originality, and visualization coverage [342]. They also report surprising discoveries, such as the use of the system in ways other than the intended purpose.

Research Agenda: While a plethora of such system-level evaluations exist in the area of visual analytics and visualization, very few of those explicitly incorporate aspects of progressivity. Rich and qualitative descriptions from such case and design studies [44] will help to uncover previously unexplored facets in the realm of PDA systems. They will help us to better characterize and formalize metrics of success, but also to learn more about the utility and usability characteristics that PDA solutions will need to offer if they are to be fully integrated into daily working practices.

9.3 Summary

PDA has introduced two new dimensions for evaluation: *trading accuracy/completeness for time* and *balancing stability and quality*. These criteria will need to be well-defined to be integrated into the existing evaluation frameworks of progressive systems. To improve the evaluation of progressive systems, we also need a better separation of concerns among the multiple parts of the progressive pipeline: data management, algorithm or machine learning (if relevant), and visualization and interaction. Right now, these parts are difficult to separate because of the integrated nature of progressive systems. This makes it difficult to design experiments targeting just one part rather than the whole system. To separate them, we would need a better understanding of the requirements and guarantees from one part to the next. In particular, algorithms should be able to let users know they can trade better quality for less stability, but interactive interventions in algorithms are not currently considered in algorithm evaluation, and visualization evaluation does not consider algorithm interventions in their evaluation. **Research Agenda:** More research will be needed to enable communication between algorithms and visualizations for control of the stability/quality trade-off, especially with a view to discovering meaningful evaluation criteria. For example, evaluating a progressive visualization would need a characterization of its possible requirements in terms of progressive results convergence and pace for the parts coming upstream in the pipeline. Progressive visualization techniques could have constraints on how progress feedback and uncertainty should be available to match its encoding. In turn, evaluating data management and machine learning techniques would require a characterization of steerability capabilities or interface requirements, describing how a downstream interface might have control over loading speed parameters or filtering queries progressively.

Research Agenda: Additionally, there is a need for more empirical studies comparing progressive systems and traditional systems in different tasks and domain applications with a view to identifying potential perceptual and cognitive bias and developing design guidelines to help progressive systems mitigate them. As progressive systems become more widely available for different applications, it will also become more important to evaluate the adoption of PDA systems. Finally, it will be critical to orchestrate the various forms of evaluation into a larger framework for PDA evaluation. For the areas of visualization and visual analytics, researchers have provided such frameworks, for instance, as nested models [196, 211] and as guiding scenarios [137, 168]. These frameworks help researchers pick adequate evaluation methods and methodologies throughout the entire visualization design and research process. PDA will necessitate reflecting on, adapting, and extending these models.

10 Challenges and Research Agenda

This book has outlined the state of the art in Progressive Data Analysis research. Throughout, it has highlighted challenges—open questions that have yet to be resolved, and opportunities where new algorithms, techniques, or technologies could open the way to greater adoption of Progressive Data Analysis.

This chapter consolidates some of the cross-cutting themes and brings together broader questions. The goal is to provide a research agenda for Progressive Data Analysis that could lead to the broad adoption of these techniques. Ultimately, of course, it would be desirable for off-the-shelf PDA systems to be available as industrial-strength systems to bring these benefits to end users.

We begin with a discussion of the overarching challenges in the field—questions that cut across multiple topics and require new thinking about how to build infrastructures. We follow by describing a selection of high-risk / high-value challenges that, while difficult, have the potential to dramatically reshape the field. We then conclude with a selection of other research challenges from the previous chapters.

10.1 Overarching Challenges to Broader Adoption of PDA

An analytics system can be thought of as a pipeline: data starts from a data source and, through a series of computation algorithms, flows through the system, coming out on the other end as a user experience and visualization, and ultimately, as a final result and report. While this pipeline metaphor is common in data analysis, it is true in a more literal way in the case of Progressive Data Analysis: a system has to take into account the continuous (or repeated) flow of data through the pipeline stages. This means that optimally, the database, the algorithms, and the visualization should be designed together: each must be aware that the other components may be part of a progressive system and comply with its requirements (Table 2.1).

Conversely, in a steerable system, the user input must flow backward through the system to drive different computations.

This gives rise to a number of challenges:

- Building an infrastructure to manage this pipeline, while providing consistent mechanisms throughout

- Ensuring that databases, algorithms, and user experiences are all aware of the concepts of progressivity and uncertainty
- Making sure that algorithms and user experiences are aware of the concept of stability, and the trade-offs between stability and quality (Section 5.2.1)
- Designing system infrastructure that allows for reproducible computations, so that the same run will produce the same results in the same order when needed (Section 9.2.2)
- Evaluation of progressive systems, algorithms, and visualizations should take into account at least three different deadlines: the first meaningful progressive result, the earliest result accurate for decision-making, and the total computation time if needed (Section 9.1.2)
- Steering should also be considered for evaluation, even though there is currently no standard methodology for taking that into account
- Most scalable eager exploratory systems assume that data has been loaded locally and, in some cases, indexed in a database before it can be used for exploration (e.g., Falcon [206], Mosaic [114], Cartolabe [39], Nanocubes [177]). Progressive systems have the potential to cold-start an exploration. The assumptions and the preprocessing time (i.e., overall time from intent to first *meaningful* results as mentioned in Section 9.1.2), should be clarified to allow fair comparisons between exploratory systems and supporting algorithms [249].

10.2 Data Management

- Databases often prefer data stored in an efficient, ordered format. By contrast, progressive algorithms prefer shuffled data to ensure fair sampling. Both arbitrary access to the database and random selection of rows are expensive and are slow operations that can act as a bottleneck to efficient Progressive Data Analysis systems. It would be helpful to find new ways to ensure that data can be efficiently shuffled for real applications (Section 3.5.1).
- More research is needed to identify and categorize sampling strategies that can be used by progressive algorithms, as well as to characterize their efficiency, especially for large-scale distributed storage.
- If data is accessed through a database, it would be helpful to provide a mechanism to read it by chunk in random order (shuffled) to improve the quality of progressive algorithms down the pipeline. OLA-RAW [50] (see Section 3.5.1) shuffles the data incrementally using sophisticated mechanisms to avoid global

pre-processing. While most databases can sort the data (at high cost), very few are able to shuffle data. Although the problem has been explored in research [237], few solutions are available with existing systems, and those are not readily applicable to PDA.

- Modern data management systems rely on well-specified SQL standard(s) and novel JSON-based formats and APIs. Unfortunately, these standards are not applicable to PDA, as they assume eager, one-shot function calls. New conventions and standards should be designed to allow progressive data systems to interoperate in a standard way with the rest of the PDA pipeline.
- More experience is needed to understand the level of support required between the progressive database and the analytical operations down the PDA pipeline to avoid redundant work, while releasing the analytical front-end from performing operations, which are more cost-effective to do in the database.
- Query steering is important for exploration in PDA, but is far from standard in data management systems. More research is needed on adapting existing systems or designing new systems for query steering, as well as on standardizing the information to communicate through an API to process the results of this steering downstream.
- More progressive services are needed to use PDA over the network for exploring large databases. These services should support important operations, including progressive (min, max) for a specified query, approximate attribute distributions, approximate distinct-values counts, approximate quantiles, and possibly different levels of shuffling to trade speed for quality.

10.3 Data Structures and Algorithms

As explained in Chapter 4, orchestrating a general PDA application calls for consistent mechanisms and requirements at the system level and at each level of libraries for scientific computing. This consistency is currently difficult to achieve because a PDA pipeline encompasses components designed by different scientific and technical domains, whereas consistency is required between all the domains.

The challenges for data structures and algorithms are:

- More research is needed to find general strategies, when possible, to adapt iterative algorithms to be applicable to data arriving in chunks.
- Ideally, all progressive algorithms should be able to handle data growing by chunks, but this can come at a high cost. Therefore, it also makes sense for

progressive libraries to provide multiple variations of the same algorithm that are better suited to data arriving in chunks or all at once.

- More generally, solutions can be explored, with different trade-offs, to support the progressive implementation of important algorithms.
- The same is true for steering: most fundamental algorithms are not designed to deal with data changing dynamically.
- A specific compiler is needed to transform a general set of linear algebra operations written in conventional notation into an efficient progressive implementation, performing matrix and vector operations in an appropriate order (e.g., row-wise or column-wise) to allow continuously generating meaningful partial results.
- Generalizing progressive data analysis to networks is still an open problem. Many graph/network algorithms are based on visitors that are iterative but do not always deliver useful partial results. For example, computing the shortest path in a graph uses an iterative algorithm (e.g., Dijkstra's) and can be interrupted at any time, but the partial results are not always solutions, whereas progressive algorithms are expected to deliver intermediate solutions. Could such graph algorithms be adapted to provide useful or meaningful partial results?
- More research will be needed to adapt existing data sketching methods to handle general progressive operations efficiently. They need two adaptations: bounding their computation time (latency) and, for steerable progressivity, dealing with deletions or value updates in a smarter manner than complete recomputation.
- More work is needed to characterize existing algorithms regarding their ability to become progressive, to help in finding generic transformation methods or strategies to make more of them progressive.

10.4 Visualization

Progressive Data Analysis offers new opportunities for data visualization. Chapter 5 outlines many of the specialized needs for visualization in Progressive Data Analysis systems. Some other interesting research challenges include:

- We lack good representations for scalability in visualizations. While some visualizations are inherently based on aggregations, and thus are scalable "for free", we are also accustomed to seeing visualizations on an individual data level. We should articulate scalable versions of classic visualizations — like a scalable line chart with many data points or millions of lines or area charts with multiple dimensions. We should carefully articulate which dimensions can take extreme scalability (e.g., number of points represented), and which need to be constrained to reasonable numbers (e.g., number of series).

- We need to learn more about how to manage stable visualizations as progressive information arrives. Some algorithms are intrinsically stable for certain types of changes and datasets. For other visualization algorithms, we can find stabilized variants. For others, we might be able to trade off stability and other measures of quality. (For example, a stable treemap might get increasingly bad aspect ratios on its rectangles). In those algorithms, we may want to allow the user to interactively trigger "repairing" the visualization, explicitly breaking the stability to achieve better quality, as mentioned by Jo *et al.* [142]. This strategy has implications for the whole progressive system since algorithms should inform the user that repair actions are available. Changing the visualization layout certainly breaks the user's mental map and would require mechanisms to help users recover it.

Designing interfaces for Progressive Visual Analytics will also drive a need to create ways to communicate with users—a set of guidelines, conventions, and controls that we refer to here as a "design language." We will need to learn how to communicate uncertainty, progress, and partial results to users. We will need to design steerable interfaces that can help users understand what changes to expect.

We see opportunities for new design languages in each of the following:

- Visualizing uncertainty on graphs, especially as it changes with additional progress (but see also Section 10.5).
- Selecting the best visual encoding for a progressive visualization and which features it has to provide.
- Using the user's interactions to direct visualization and selecting controls to best help steer visualization and computation in a progressive system.
- Choosing controls that appropriately cue the user for proper responsiveness and computation expectations (see Section 7.1).

Overall, more research should be conducted on the advantages, disadvantages, and design patterns to choose where progression and quality indicators should be shown in the UI and visualizations.A systematic approach is needed to identify the design space of choices in controlling and interacting with PVA. (See also Section 5.5).

10.5 Quality and Uncertainty

Fundamental to the concept of allowing users to explore the results of progressive analytics is giving them the ability to interpret their partial and progressive results. In Chapter 6, we outlined contemporary issues in visualizing uncertainty and progressive uncertainty. Some of the major issues in uncertainty include:

- Modeling uncertainty, quality, and progression in a consistent way across the whole PDA pipeline to help combine and visualize them.
- When uncertainty is too complex to be combined across pipeline stages, a consistent way of using the Monte Carlo method is designed to collect ensemble uncertainty from full computation pipelines.
- Improve methods for computing the uncertainty levels algorithmically, including finding uncertainty guarantees and good approximations.
- Choosing good representations for uncertainty that allow users to interpret both the uncertainty and the quality of the estimates.
- Carry out perceptual and psychological research to gain a better understanding of how users make decisions under progressive uncertainty.
- Uncertainty comes in several parts: a computation might have measures of both local (per-element) and global uncertainty; both of these may be proxies for quality measures. We need to research design languages to help better communicate the different sorts of uncertainty.
- It is unclear how to compute or communicate uncertainty for high-dimensional data analysis and complex estimators. Only a small set of partial measures have been found— for example, in a *k*-means model, the amount of movement in the centroids can serve as a proxy for quality.
- Some quality measures take longer to compute than the progressive result. When designing systems, we could encounter a situation where the quality measure could either slow down the main visualization or appear with a delay. How should we help users interpret such a situation?
- Distribution-based probability methods, such as Monte Carlo estimation, may create different models of uncertainty. There are opportunities to learn new ways to visualize them.
- More work is needed to study uncertainty in the context of machine learning progressive pipelines, where the uncertainty at the end of the pipeline depends on choices made all through the pipeline but only shows up late in the computation after the model has learned for long enough.

10.6 Human Aspects

We turn to users themselves to examine the human implications of progressive visualization. The taxonomy of task and user types described in Chapter 7) presents new research challenges.

- We need to build a design language that helps clarify to users what the capabilities each of their roles holds.
- We need to understand how users move between these roles, and under what circumstances, during the analysis process.
- We need to learn how analytical tasks in Progressive Visual Analytics relate to their more traditional counterparts, to find analogues to guidelines like Shneiderman's Visual Information-Seeking Mantra [282], and to expand traditional task typologies and taxonomies to progressive workflows.
- Explainable approaches are needed to shed light on the progressive usage and training of black-box models.
- We need to find ways for PVA systems to mitigate various forms of bias Section 7.5.
- We need to learn the appropriate requirements for interactivity, accuracy, and timing. When does it make sense to produce a more refined output compared to producing responses to users? What sorts of users will tolerate a PDA system that takes longer than a baseline system to reach a final response, but can provide partial results much more quickly?
- The study of collaborative PDA solutions is another research challenge that largely remains open.

10.7 Machine Learning

We now turn to the challenges for progressive methods in Machine Learning (ML), as discussed in Chapter 8. While in an ideal world, ML model building and usage would be fully automatic, both processes are in reality tied to human experts/users interacting with them. Given that datasets tend to be large in ML, progressive methods offer a viable opportunity to make this interaction more efficient and effective by letting users steer the respective algorithms on the fly. However, more research is needed to overcome the inherent challenges of making ML a progressive process.

- Many fundamental ML algorithms (beyond kNN search) may be reformulated as progressive versions so that they can be used in a PDA pipeline. These algorithms need to be able to work incrementally with partial data, be robust in terms of data stationarity, and ensure proper convergence rates. In addition, issues of quality and uncertainty are key in establishing a good interplay between users and the algorithms. Work in online learning could provide interesting starting points for such algorithms.

- Training very large deep neural networks is slow and error-prone. Progressive methods could help to support monitoring and steering for this process. However, integrating PDA into deep-learning frameworks, such as Torch or Keras, is an open problem.
- While methods for reinforcement learning through human feedback (RLHF) are progressive by design, investigating how to steer and verify their feedback process is an open research question.
- To support progressive model building, we need informative and consistent progress and performance indicators that address the inherent uncertainties of progressive techniques.
- We need more advanced methods of "active learning" that can elicit complex tacit knowledge from people to facilitate effective steering of models in progressive learning settings.
- With regard to designing effective progressive ML techniques, it is also imperative to gain a better understanding of human factors (see also Section 10.6), e.g., by conducting "in the wild" studies (see also Section 10.8).

10.8 Evaluation

As discussed in Chapter 9, the new PDA paradigm will also substantially impact how we evaluate these analytical systems. The core difference between PDA and classical eager systems is the *inherent trade-off between time and quality/confidence*. This central trade-off needs to be reflected in the evaluation of PDA approaches along the entire pipeline. Not only are new evaluation methods and methodologies needed that can foster this inherent trade-off, but more studies will also be needed to better understand PDA and how it compares to traditional approaches.

In terms of new and adapted methods, methodologies, and metrics, we are facing the following challenges:

- The overarching challenge of adopting existing evaluation methods and methodologies from the visualization and database communities, and extending them so that they account for the new requirements of PDA systems. A new methodology is needed that combines different methods (quantitative and qualitative, objective and subjective) along the entire PDA pipeline.

- There already exist a plethora of objective measures available for the quantitative evaluation of analytical systems. An important challenge will be to characterize when to use which measures, for what purpose, under what condition/scenario, and how to weigh the trade-off between them. Many of the old tasks and measures will also remain valid for PDA. However, at the same time, new representative tasks and measures will need to be devised that account for the time/quality trade-off that is inherent to PDA. Measures to address this trade-off need to account for aspects such as early-stopping [52], steering of the progressive process, and sometimes the stability of the intermediary results.
- Another important characteristic of evaluating PDA systems is the intrinsic interdependence of human and computational factors. For instance, how can we quantitatively evaluate and compare approaches while involving human interventions? New evaluation methods are needed that tightly integrate and bridge these two worlds, given the specific requirements of PDA.
- Assessing whether a progressive system meets the requirements listed in Table 2.1 will require that their authors, and ultimately the PDA community, clarify these concepts, in addition to stating that their system implements them fully, partially, or given specific definitions. Properly articulating assumptions is critical for a faithful evaluation of PDA approaches, and will be invaluable in devising a higher-level framework for PDA evaluation.
- Methodologies are needed to describe decision-making requirements, sensemaking strategies, and assess the optimality of user behaviors. Part of a system's evaluation would also be to determine how well it performs under different requirements (e.g., fast decision, or minimum accuracy).
- New, better, standardized benchmarks are needed with both representative data distributions and query tasks.

So far, only a few empirical evaluations of approaches along the PDA pipeline exist. More work will be needed that helps to empirically characterize, assess, and compare the benefits and drawbacks of different approaches:

- More studies are needed that characterize the differences between traditional and PDA approaches, and provide guidance on when to use which.
- PDA interfaces will bring a new understanding of how data is visualized, understood, and leveraged by users in their analysis strategies. One important challenge, for instance, regards the update rate: How often should changes be shown in order not to either overwhelm the user with too frequent changes or

fail to maintain their attention with too infrequent changes? PDA interfaces pose an inherent trade-off between stability (only small visual changes) vs. accuracy (accurate representation at each time), and human-subject studies are needed to devise different strategies for dealing with this and other trade-offs in the user interface.

- Careful and rigorous perceptual and cognitive studies will be needed for various aspects of the PDA pipeline. These include aspects such as change blindness, latency perception, and cognitive biases. How exactly do such perceptual and cognitive aspects appear in progressive setups, do they cause new problems beyond traditional systems, and how well do countermeasures work?
- Progressive machine learning has been evaluated at the algorithm level and the system level. With the advent of explainable AI, they should also be evaluated regarding their capability of explaining or adding transparency to the learning process.
- Longitudinal case, design, and field studies "in the wild" will help to qualitatively understand the implications of PDA on real-world analytical practices and how end users will adopt them.
- PDA is a highly faceted research area. Studies will need to focus on specific aspects, and will usually not be able to cover the full evaluation breadth along the entire pipeline. A proper description of the focus and its limitations will help build up larger-scale models based on different individual studies.
- Last but not least, many studies will be needed to understand the implications of new and refined evaluation methods and methodologies as discussed above. For instance, the trade-offs between different measures of user performance are dependent on the particular context of user experiments for progressive visualization and will differ from each other as well as from current evaluation practices. Reflecting on PDA evaluation experiences can help others in similar endeavors.

10.9 Threats and Opportunities

To understand the state of the evolution of PDA, we can refer to the work of Brian Gaines [95] and his BRETAM model. Gaines argues that new knowledge or products in science and technology evolve in stages, which begins with a ***b**reakthrough*, when the knowledge or artifact is presented to a community. This then requires a ***r**eplication* stage during which the community tries to replicate them with variations, followed by an ***e**mpiricism* stage when success and failures improve the knowledge/product. This leads to the development of ***t**heories*, ***a**utomation* and, eventually, to ***m**aturity*, at which point the knowledge/product is assimilated and used without question. Without a doubt, at the time this book is published, PDA is at the early *replication* stage, although with a few specific issues it is sometimes in the *empiricism* stage. In the past, some technologies have taken many years to get past the replication stage. Greenberg mentions as food for thought and inspiration the groupware toolkits in [104]:

> The big catch is that because groupware is hard to build, we have essentially throttled its development at the replication stage. This has minimized product invention and innovation as well as hands-on experience to all but the CSCW [Computer Supported Collaborative Work] research community and a few well-resourced developers. Thus, the necessary creativity leading to product evolution was stifled.

Nowadays, groupware tools are assimilated and used without question, but it took fifteen years to go beyond the replication stage. The seminal article on Online Aggregation [117] was published in 1997. More than twenty five years have passed and it has still not been adopted. So, why has Online Aggregation still not been able to break through? Is it because we still need to build a successful complete system, or are there deeper challenges to adoption? What would it take to show that PDA is actually a Terrible Idea? The authors of this book believe, on the contrary, that it is a promising idea that should be pursued—until it grows beyond the replication stage and eventually reaches maturity.

Another important point that is worth mentioning, although so far only based on anecdotal evidence, is the ability of progressive visualizations to reveal the inner workings of some algorithms and phenomena. For example, Li *et al.* report that "By visualizing the dynamics of the t-SNE computation, we identified smooth phenotypic transitions from cells within the [. . .] cluster to both the [. . .], revealing potential differentiation trajectories." [174]. Following the unfolding of a complex algorithm progressively might convey the way it operates and add transparency, as well as some level of explainability to data analysis. It may also reveal data characteristics only visible through the animation of a progressive algorithm. More work is needed to understand where, when, and why these properties of PDA are effective.

11 Conclusion

The starting point of this book is the glaring fact that current data exploration and visualization systems are severely limited in scalability because they require low latency, and this latency is currently tied to computation time. This time grows unbounded with data size and the complexity of data analysis algorithms; modern data sizes and algorithm complexity involve compute times that easily exceed human limits. Without a solution, the existing data exploration and visualization systems might become unusable for data analysts who routinely need to analyze terabytes of data using sophisticated analytical algorithms.

In this book, we explained how Progressive Data Analysis addresses this scalability problem by decoupling response time (latency) from computation time, and why it, therefore, could become an important paradigm to support data exploration at scale in the near future. However, while addressing the latency issue, this new paradigm introduces many new and complex challenges that need to be addressed before PDA can become usable and mainstream.

We have explained what PDA is and how it works at different levels of the data analysis and exploration pipeline, from data management to the human analysts who need to use it. Along the way, we listed several exciting challenges and opportunities to develop PDA and turn it into a mainstream paradigm. We also discussed the applicability of PDA for real applications.

We believe that, in Gaines's framework [95] (see Section 10.9), PDA remains stuck in the replication stage, and the purpose of this book is to unlock the paradigm and provide a clear roadmap to replication, empiricism, and the stages beyond. Passing the replication stage will require joining efforts between separate disciplines instead of working in silos, as academic research tends to encourage. We hope many students will be inspired by the multiple cross-disciplinary challenges raised in this book to contribute to the development of PDA.

Although the vision of this book is to achieve fully progressive systems, partially progressive systems are also possible and certainly suitable for important purposes. More work will be needed to understand how progressive and non-progressive systems can live together and support data exploration at scale in a wide variety of contexts and setups. Whether fully progressive or partially progressive, they can be transformational for data exploration, as well as for helping machine learning and complex simulations with humans in the loop.

Over the last decade, data-driven approaches have fundamentally changed the world around us, and opened door to tremendous advances in science, technology, and society. Many of these advances are now subsumed under the umbrella of Artificial Intelligence (AI). A core pillar of their success can be largely attributed to the substantially increased amount and size of data underlying these models. Similarly important, however, is the fact that human experts, scientists, and engineers built and tested these models. To this end, it has been and will continue to be extremely important that humans are able to properly visualize, interact with, and understand the data at hand. Or as Michael Jordan, distinguished professor for statistics at Berkeley, puts it: *"We will need well-thought-out interactions of humans and computers to solve our most pressing problems"* [148]. To this end, we believe that methods for progressive data analysis will become ever more essential as a way of dealing with the continuous exponential growth of data. While challenges remain, PDA will eventually allow us to interactively work with, analyze, and understand data at scales that are not yet feasible today.

Bibliography

[1] S. Afzal, R. Maciejewski, and D. S. Ebert. Visual analytics decision support environment for epidemic modeling and response evaluation. In *Proceedings of IEEE VAST*, pages 191–200. IEEE, 2011. doi:10.1109/VAST.2011.6102457. 113

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proc. of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42. ACM, 2013. doi:10.1145/2465351.2465355. 34

[3] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. NoDB: Efficient Query Execution on Raw Data Files. In *International Conference on Management of Data*, SIGMOD '12, page 241–252, New York, NY, USA, 2012. ACM. doi:10.1145/2213836.2213864. 34

[4] F. Alam, F. Ofli, and M. Imran. Processing Social Media Images by Combining Human and Machine Computing during Crises. *International Journal of Human-Computer Interaction*, 34(4):311–327, 2018. doi:10.1080/10447318.2018.1427831. 125

[5] F. Alam, F. Ofli, M. Imran, and M. Aupetit. A Twitter Tale of Three Hurricanes: Harvey, Irma, and Maria. In *International Conference on Information Systems for Crisis Response and Management (ISCRAM)*, 2018. 124

[6] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003. doi:10.1007/s10107-003-0436-0. 24

[7] G. Alicioglu and B. Sun. A survey of visual analytics for Explainable Artificial Intelligence methods. *Computers & Graphics*, 102:502–520, 2022. doi:10.1016/j.cag.2021.09.002. 119

[8] M. Angelini, T. May, G. Santucci, and H.-J. Schulz. On Quality Indicators for Progressive Visual Analytics. In *Proc. of the 10th International EuroVis Workshop on Visual Analytics (EuroVA'19)*, pages 25–29. Eurographics Association, 2019. doi:10.2312/eurova.20191120. 92, 93, 100, 112, 115

[9] M. Angelini and G. Santucci. Modeling Incremental Visualizations. In *EuroVis Workshop on Visual Analytics*. The Eurographics Association, 2013. doi:10.2312/PE.EuroVAST.EuroVA13.013-017. 22, 73, 80

[10] M. Angelini and G. Santucci. On Visual Stability and Visual Consistency for Progressive Visual Analytics. In *Proceedings of IVAPP*, pages 335–341. SciTePress, 2017. doi:10.5220/0006269703350341. 112, 115, 162

[11] M. Angelini, G. Santucci, H. Schumann, and H.-J. Schulz. A Review and Characterization of Progressive Visual Analytics. *Informatics*, 5(3), 2018.

doi:10.3390/informatics5030031. 10, 22, 80, 119, 121, 123

[12] Apache Arrow. https://arrow.apache.org/. Accessed: 2024-01-06. 62

[13] M. Aumüller, E. Bernhardsson, and A. Faithfull. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020. doi:10.1016/j.is.2019.02.006. 60

[14] M. Aupetit and M. Imran. Interactive Monitoring of Critical Situational Information on Social Media. In *International Conference on Information Systems for Crisis Response And Management (ISCRAM)*, pages 673–683, 2017. 124, 125, 127

[15] H. Avron, M. Kapralov, C. Musco, C. Musco, A. Velingker, and A. Zandieh. Random Fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *International Conference on Machine Learning*, pages 253–262. PMLR, 2017. 142

[16] S. K. Badam, N. Elmqvist, and J.-D. Fekete. Steering the Craft: UI Elements and Visualizations for Supporting Progressive Visual Analytics. *Computer Graphics Forum*, 36(3):491–502, 2017. doi:10.1111/cgf.13205. 19, 22, 78, 79, 80, 82, 85, 86, 90, 95, 97, 104, 105, 106, 112, 119, 121, 123, 159, 160, 162, 168, 210

[17] J. Baron. *Thinking and deciding*. Cambridge University Press, 2000. doi:10.1017/CBO9780511840265. 123

[18] J. Baron, J. Beattie, and J. C. Hershey. Heuristics and biases in diagnostic reasoning: II. Congruence, information, and certainty. *Org. Behavior and Human Decision Processes*, 42(1):88–110, 1988. doi:10.1016/0749-5978(88)90021-0. 123

[19] L. Battle, M. Angelini, C. Binnig, T. Catarci, P. Eichmann, J. Fekete, G. Santucci, M. Sedlmair, and W. Willett. Evaluating Visual Data Analysis Systems: A Discussion Report. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 4:1–4:6, 2018. 154

[20] L. Battle, R. J. Crouser, A. Nakeshimana, A. Montoly, R. Chang, and M. Stonebraker. The role of latency and task complexity in predicting visual search behavior. *IEEE Trans. Vis. Comput. Graph.*, 26(1):1246–1255, 2020. doi:10.1109/TVCG.2019.2934556. 164

[21] L. Battle, P. Eichmann, M. Angelini, T. Catarci, G. Santucci, Y. Zheng, C. Binnig, J.-D. Fekete, and D. Moritz. Database Benchmarking for Supporting Real-Time Interactive Querying of Large Data. In *International Conference on Management of Data*, volume 17 of *SIGMOD ’20*, Portland, OR, United States, June 2020. ACM. doi:10.1145/3318464.3389732. 34, 154

[22] Y. Bengio. *Learning deep architectures for AI.* Now Publishers Inc, 2009. 138, 143

[23] L. Berg, T. Ziegler, C. Binnig, and U. Röhm. ProgressiveDB: Progressive Data Analytics as a Middleware. *Proc. VLDB Endow.*, 12(12):1814–1817, Aug. 2019. doi:10.14778/3352063.3352073. 42, 45, 150

[24] J. Bernard, M. Zeppelzauer, M. Sedlmair, and W. Aigner. VIAL: a unified process for visual interactive labeling. *The Visual Computer*, 34(9):1189–1207, 2018.

doi:10.1007/s00371-018-1500-3. 146

[25] E. Bertini and G. Santucci. Give Chance a Chance: Modeling Density to Enhance Scatter Plot Quality through Random Data Sampling. *Information Visualization*, 5(2):95–110, 2006. doi:10.1057/palgrave.ivs.9500122. 70, 72

[26] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, Jan. 2017. doi:10.1137/141000671. 49

[27] A. Bezerianos, P. Dragicevic, and R. Balakrishnan. Mnemonic Rendering: An Image-Based Approach for Exposing Hidden Changes in Dynamic Displays. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, page 159–168, New York, NY, USA, 2006. ACM. doi:10.1145/1166253.1166279. 163

[28] P. Billingsley. *Convergence of probability measures*. Wiley Series in Probability and Statistics: Probability and Statistics. John Wiley & Sons Inc., New York, second edition, 1999. A Wiley-Interscience Publication. 21

[29] P. A. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Conference on Innovative Data Systems Research, CIDR*, volume 5, pages 225–237, 2005. 55

[30] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998. 156, 158

[31] L. Bottou. Online Algorithms and Stochastic Approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998. URL `http://leon.bottou.org/papers/bottou-98x`. revised, oct 2012. 136

[32] E. H. Bowman. Consistency and optimality in managerial decision making. *Management Science*, 9(2):310–321, 1963. 165

[33] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike-20&path=ASIN/0521833787`. 140

[34] U. Brandes. Force-Directed Graph Drawing. In *Encyclopedia of Algorithms*, pages 768–773. Springer, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_648. 57

[35] M. Brehmer and T. Munzner. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2376–2385, 2013. doi:10.1109/TVCG.2013.124. 114, 128

[36] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. doi:10.1023/A:1010933404324. 137

[37] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners, 2020,

2005.14165. 134

[38] J. S. Bruner and M. C. Potter. Interference in Visual Recognition. *Science*, 144(3617):424–425, 1964. doi:10.1126/science.144.3617.424. 164

[39] P. Caillou, J. Renault, J.-D. Fekete, A.-C. Letournel, and M. Sebag. Cartolabe: A Web-Based Scalable Visualization of Large Document Collections. *IEEE Comput. Graph. Appl. Mag*, 41(2):76–88, 2021. doi:10.1109/MCG.2020.3033401. 9, 172

[40] M. A. Cameron, R. Power, B. Robinson, and J. Yin. Emergency Situation Awareness from Twitter for Crisis Management. In *International Conference on World Wide Web*, WWW'12, pages 695–698. ACM, 2012. doi:10.1145/2187980.2188183. 124

[41] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. 13, 164, 210

[42] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proceedings of CHI*, CHI '91, page 181–186, New York, NY, USA, 1991. ACM. doi:10.1145/108844.108874. 108, 110

[43] H. Cardot and D. Degras. Online principal component analysis in high dimension: Which algorithm to choose? *International Statistical Review*, 86(1):29–50, 2018. doi:10.1111/insr.12220. 60

[44] S. Carpendale. Evaluating information visualizations. In *Information visualization*, pages 19–45. Springer, 2008. 168

[45] C. Carpineto, S. Osiundefinedski, G. Romano, and D. Weiss. A Survey of Web Clustering Engines. *ACM Comput. Surv.*, 41(3), July 2009. doi:10.1145/1541880.1541884. 117

[46] CarrotSearch Clustering and Visualization of search results. `https://carrotsearch.com/`. Accessed: 2020-03-08. 117

[47] S. Chambi, D. Lemire, O. Kaser, and R. Godin. Better Bitmap Performance with Roaring Bitmaps. *Softw. Pract. Exper.*, 46(5):709–719, May 2016. doi:10.1002/spe.2325. 67

[48] X. Chen, J. Zhang, C.-W. Fu, J.-D. Fekete, and Y. Wang. Pyramid-based Scatterplots Sampling for Progressive and Streaming Data Visualization. *IEEE Trans. Vis. Comput. Graph.*, 28(1):593–603, 2022. doi:10.1109/TVCG.2021.3114880. 72, 156, 157

[49] Y. Cheng and F. Rusu. Parallel In-Situ Data Processing with Speculative Loading. In *International Conference on Management of Data*, SIGMOD '14, page 1287–1298, New York, NY, USA, 2014. ACM. doi:10.1145/2588555.2593673. 34

[50] Y. Cheng, W. Zhao, and F. Rusu. Bi-Level Online Aggregation on Raw Data. In *International Conference on Scientific and Statistical Database Management*, SSDBM '17. ACM, 2017. doi:10.1145/3085504.3085514. 42, 62, 155, 172

[51] J. Choo, C. Lee, H. Kim, H. Lee, C. Reddy, B. Drake, and H. Park. PIVE: Per-Iteration visualization environment for supporting real-time interactions with computational methods. In *Visual Analytics Science and Technology (VAST), 2014 IEEE Conference*

on, pages 241–242, Oct 2014. doi:10.1109/VAST.2014.7042510. 162

[52] Y. S. Chow. *Great expectations: The theory of optimal stopping*. Houghton Mifflin, Jan. 1971. 1, 94, 101, 152, 158, 179

[53] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30:4299–4307, 2017. 144

[54] J. D. Cohen. Drawing Graphs to Convey Proximity: An Incremental Arrangement Method. *ACM Trans. Comput.-Hum. Interact.*, 4(3):197–229, sep 1997. doi:10.1145/264645.264657. 73

[55] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996. doi:10.1613/jair.295. 146

[56] A. Collette. *Python and HDF5*. O'Reilly, 2013. 55

[57] G. Cormode. Data Sketching. *Commun. ACM*, 60(9):48–55, Aug. 2017. doi:10.1145/3080008. 52, 97

[58] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends Databases*, 4(1–3):1–294, Jan. 2012. doi:10.1561/1900000004. 44, 52, 61

[59] M. Correll and M. Gleicher. Error bars considered harmful: Exploring alternate encodings for mean and error. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2142–2151, 2014. doi:10.1109/TVCG.2014.2346298. 88

[60] M. Correll, D. Moritz, and J. Heer. Value-Suppressing Uncertainty Palettes. In *Proceedings of CHI*, CHI '18, pages 1–11. ACM, 2018. doi:10.1145/3173574.3174216. 77, 82, 88

[61] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sept. 1995. doi:10.1007/bf00994018. 138, 140, 141

[62] N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy. Optimal transport for domain adaptation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(9):1853–1865, 2017. doi:10.1109/TPAMI.2016.2615921. 145

[63] A. Crisan and B. Fiore-Gartland. Fits and Starts: Enterprise Use of AutoML and The Role of humans in the loop. In *Proceedings of CHI*, CHI '21, pages 601:1–601:15, 2021. doi:10.1145/3411764.3445775. 137

[64] Crisis Computing, Dec 2019. URL `https://crisiscomputing.qcri.org/`. 124

[65] R. J. Crutcher and A. F. Healy. Cognitive operations and the generation effect. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15(4):669–675, 1989. doi:10.1037/0278-7393.15.4.669. 131

[66] Z. Cui, J. Kancherla, H. C. Bravo, and N. Elmqvist. Sherpa: Leveraging User Attention for Computational Steering in Visual Analytics. In *Proceedings of the Visualization in Data Science (VDS) Symposium*, pages 48–57. IEEE, 2019. doi:10.1109/VDS48975.2019.8973384. 113, 116, 118

[67] E. Dimara, G. Bailly, A. Bezerianos, and S. Franconeri. Mitigating the Attraction

Effect with Visualizations. *IEEE Trans. Vis. Comput. Graph.*, 25(1):850–860, Jan. 2019. doi:10.1109/TVCG.2018.2865233. 165

[68] E. Dimara, S. Franconeri, C. Plaisant, A. Bezerianos, and P. Dragicevic. A Task-Based Taxonomy of Cognitive Biases for Information Visualization. *IEEE Trans. Vis. Comput. Graph.*, 26(2):1413–1432, 2020. doi:10.1109/TVCG.2018.2872577. 165

[69] E. Dimara and C. Perin. What is Interaction for Data Visualization? *IEEE Trans. Vis. Comput. Graph.*, 26(1):119–129, Jan 2020. doi:10.1109/TVCG.2019.2934283. 108

[70] C. Domeniconi and D. Gunopulos. Incremental support vector machine construction. In *International Conference on Data Mining (ICDM)*, pages 589–592, 2001. doi:10.1109/ICDM.2001.989572. 140, 141

[71] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. Yarkhan, M. Abalenkovs, N. Bagherpour, and et al. PLASMA: Parallel Linear Algebra Software for Multicore Using OpenMP. *ACM Trans. Math. Softw.*, 45(2), May 2019. doi:10.1145/3264491. 55, 59

[72] E. P. dos Santos Amorim, E. V. Brazil, J. Daniels, P. Joia, L. G. Nonato, and M. C. Sousa. iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 53–62, Oct 2012. doi:10.1109/VAST.2012.6400489. 117

[73] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. doi:10.3138/FM57-6770-U75U-7727. 103

[74] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984. URL `https://ci.nii.ac.jp/naid/10000023432/en/`. 10, 57, 73

[75] K. Echihabi, T. Tsandilas, A. Gogolou, A. Bezerianos, and T. Palpanas. ProS: Data Series Progressive k-NN Similarity Search and Classification with Probabilistic Quality Guarantees. *The VLDB Journal*, 32(4):763–789, nov 2022. doi:10.1007/s00778-022-00771-z. 60

[76] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993. 102, 153

[77] P. Eichmann, E. Zgraggen, Z. Zhao, C. Binnig, and T. Kraska. Towards a Benchmark for Interactive Data Exploration. *IEEE Data Eng. Bull.*, 39(4):50–61, 2016. 155

[78] M. A. Elliott, C. Nothelfer, C. Xiong, and D. A. Szafir. A Design Space of Vision Science Methods for Visualization Research. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1117–1127, 2021. doi:10.1109/TVCG.2020.3029413. 163

[79] J. S. B. Evans, J. L. Barston, and P. Pollard. On the conflict between logic and belief in syllogistic reasoning. *Memory & Cognition*, 11(3):295–306, 1983. doi:10.3758/BF03196976. 120

[80] J. Farman. *Delayed Response: The Art of Waiting from the Ancient to the Instant World*. Yale University Press, 2019. 149

[81] J.-P. Farrugia and B. Péroche. A progressive rendering algorithm using an adaptive

perceptually based image metric. *Computer Graphics Forum*, 23(3):605–614, 2004. doi:10.1111/j.1467-8659.2004.00792.x. 159

[82] J.-D. Fekete, D. Fisher, A. Nandi, and M. Sedlmair. Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). *Dagstuhl Reports*, 8(10):1–40, 2019. doi:10.4230/DagRep.8.10.1. 11

[83] J.-D. Fekete and R. Primet. Progressive Analytics: A Computation Paradigm for Exploratory Data Analysis. *arXiv preprint arXiv:1607.05162*, abs/1607.05162, 2016. doi:10.48550/arXiv.1607.05162. 64, 65, 82, 105, 106, 167

[84] M. Fernandes, L. Walls, S. Munson, J. Hullman, and M. Kay. Uncertainty displays using quantile dotplots or cdfs improve transit decision-making. In *Proceedings of CHI*, CHI '18, pages 1–12. ACM, 2018. doi:10.1145/3173574.3173718. 106

[85] N. Ferreira, D. Fisher, and A. C. König. Sample-oriented task-driven visualizations: allowing users to make better, more confident decisions. In *Proceedings of CHI*, CHI '14, pages 571–580. ACM, 2014. doi:10.1145/2556288.2557131. 107, 114, 161

[86] F. Ferstl, K. Bürger, and R. Westermann. Streamline Variability Plots for Characterizing the Uncertainty in Vector Field Ensembles. *IEEE Trans. Vis. Comput. Graph.*, 22(1):767–776, 2016. doi:10.1109/TVCG.2015.2467204. 102

[87] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-Sklearn 2.0: The Next Generation. *arXiv preprint arXiv:2007.04074*, abs/2007.04074, 2020. doi:10.48550/arXiv.2007.04074. 166

[88] D. Fisher, S. M. Drucker, and A. C. Koenig. Exploratory Visualization Involving Incremental, Approximate Database Queries and Uncertainty. *IEEE Comput. Graph. Appl.*, 32(4):55–62, July 2012. doi:10.1109/MCG.2012.48. 52

[89] D. Fisher, I. O. Popov, S. M. Drucker, and m c schraefel. Trust Me, I'm Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster. In *Proceedings of CHI*, CHI '12, pages 1673–1682. ACM, 2012. doi:10.1145/2207676.2208294. 10, 73, 74, 82, 97, 116, 121, 167

[90] E. Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*. USAF School of Aviation Medicine, 1951. 136, 138

[91] S. Frederick, G. Loewenstein, and T. O'Donoghue. Time Discounting and Time Preference: A Critical Review. *Journal of Economic Literature*, 40(2):351–401, 2002. doi:10.1257/002205102320161311. 122

[92] S. Frey, F. Sadlo, K.-L. Ma, and T. Ertl. Interactive progressive visualization with space-time error control. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2397–2406, 2014. doi:10.1109/TVCG.2014.2346319. 10

[93] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977. doi:10.1145/355744.355745. 139

[94] T. Fujiwara, J. Chou, Shilpika, P. Xu, L. Ren, and K. Ma. An Incremental Dimensionality Reduction Method for Visualizing Streaming Multidimensional Data. *IEEE Trans.*

Vis. Comput. Graph., 26(1):418–428, 1 2020. doi:10.1109/TVCG.2019.2934433. 61, 166

[95] B. R. Gaines. Modeling and forecasting the information sciences. *Inf. Sci.*, 57-58:3–22, 1991. doi:10.1016/0020-0255(91)90066-4. 180, 182

[96] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995. 66

[97] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications)*. Springer-Verlag, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-28608-0. 52, 61

[98] K. Ghiasi-Shirazi, R. Safabakhsh, and M. Shamsi. Learning Translation Invariant Kernels for Classification. *Journal of Machine Learning Research*, 11(4), 2010. doi:10.5555/1756006.1859896. 143

[99] A. Gijsberts, R. Bohra, D. Sierra González, A. Werner, M. Nowak, B. Caputo, M. A. Roa, and C. Castellini. Stable myoelectric control of a hand prosthesis using non-linear incremental learning. *Frontiers in Neurorobotics*, 8:8, 2014. doi:10.3389/fnbot.2014.00008. 142

[100] D. T. Gilbert. How mental systems believe. *American Psychologist*, 46(2):107–119, 1991. doi:10.1037/0003-066x.46.2.107. 164

[101] D. T. Gilbert, D. S. Krull, and P. S. Malone. Unbelieving the unbelievable: Some problems in the rejection of false information. *Journal of Personality and Social Psychology*, 59(4):601–613, 1990. doi:10.1037/0022-3514.59.4.601. 164

[102] A. Gogolou, T. Tsandilas, K. Echihabi, A. Bezerianos, and T. Palpanas. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *International Conference on Management of Data*, SIGMOD ’20, pages 1857–1873. ACM, 2020. doi:10.1145/3318464.3389751. 150, 158

[103] J. Grass and S. Zilberstein. Anytime algorithm development tools. *ACM SIGART Bulletin*, 7(2):20–27, 1996. doi:10.1145/242587.242592. 27

[104] S. Greenberg. Toolkits and interface creativity. *Multim. Tools Appl.*, 32(2):139–159, 2007. doi:10.1007/S11042-006-0062-Y. 181

[105] H. Griethe and H. Schumann. The Visualization of Uncertain Data: Methods and Problems. In *Proceedings of SimVis ’06*, pages 143–156. SCS Publishing House, 2006. 94, 96

[106] P. J. Haas and J. M. Hellerstein. Ripple Joins for Online Aggregation. *SIGMOD Rec.*, 28(2):287–298, jun 1999. doi:10.1145/304181.304208. 39

[107] P. J. Haas and C. König. A Bi-Level Bernoulli Scheme for Database Sampling. In *International Conference on Management of Data*, SIGMOD ’04, page 275–286, New York, NY, USA, 2004. ACM. doi:10.1145/1007568.1007601. 43

[108] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM*

Review, 53(2):217–288, 2011. doi:10.1137/090771806. 60

[109] W. Han and H.-J. Schulz. Beyond Trust Building — Calibrating Trust in Visual Analytics. *2020 IEEE Workshop on TRust and EXpertise in Visual Analytics (TREX)*, pages 9–15, 2020. 161

[110] L. Harrison, F. Yang, S. Franconeri, and R. Chang. Ranking visualizations of correlation using Weber's law. *IEEE Trans. Vis. Comput. Graph.*, 20(12):1943–1952, 2014. doi:10.1109/TVCG.2014.2346979. 163

[111] L. Hasher, D. Goldstein, and T. Toppino. Frequency and the conference of referential validity. *Journal of Verbal Learning and Verbal Behavior*, 16(1):107–112, 1977. doi:10.1016/S0022-5371(77)80012-1. 122

[112] X. He, K. Zhao, and X. Chu. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, 212:106622, 2019. doi:10.1016/j.knosys.2020.106622. 5, 137

[113] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988. doi:10.1002/net.3230180406. 58

[114] J. Heer and D. Moritz. Mosaic: An Architecture for Scalable & Interoperable Data Views. *IEEE Trans. Vis. Comput. Graph.*, 30(1):436–446, 2024. doi:10.1109/TVCG.2023.3327189. 172

[115] J. Heinrich and D. Weiskopf. State of the Art of Parallel Coordinates. In *Eurographics 2013 - State of the Art Reports*. The Eurographics Association, 2013. doi:10.2312/conf/EG2013/stars/095-116. 71

[116] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive Data Analysis: The Control Project. *Computer*, 32(8):51–59, Aug. 1999. doi:10.1109/2.781635. 11, 24, 121

[117] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *International Conference on Management of Data*, SIGMOD '97, pages 171–182, New York, NY, USA, 1997. ACM. doi:10.1145/253260.253291. 10, 11, 24, 34, 52, 56, 73, 181

[118] R. Hema, M. Omid, and J. Rosie. Active Learning with Feedback on Both Features and Instances. *Journal of Machine Learning Research*, 7:1655–1686, 2006. 146

[119] E. G. Hetzler, V. L. Crow, D. A. Payne, and A. E. Turner. Turning the Bucket of Text into a Pipe. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 89–94. IEEE, 2005. doi:0.1109/INFOVIS.2005.34. 19, 22

[120] R. J. Heuer Jr. *Analysis of competing hypotheses*, pages 95–110. US Government Printing Office, 1999. 122

[121] R. Hipp. SQLite. `https://www.sqlite.org/`, 2000. Accessed: 2019-07-20. 56

[122] T. K. Ho. Random decision forests. In *International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995. doi:10.1109/ICDAR.1995.598994. 137

[123] M. Hogräfer, M. Angelini, G. Santucci, and H.-J. Schulz. Steering-by-example for Progressive Visual Analytics. *ACM Trans. Intell. Syst. Technol.*, 13(6):96:1–96:26,

2022. doi:10.1145/3531229. 78, 116

[124] M. Hogräfer, J. Burkhardt, and H.-J. Schulz. A Pipeline for Tailored Sampling for Progressive Visual Analytics. In *Proceedings of the 13th International EuroVis Workshop on Visual Analytics (EuroVA'22)*, pages 49–53. Eurographics Association, 2022. doi:10.2312/eurova.20221079. 31

[125] M. Hogräfer, D. Moritz, A. Perer, and H.-J. Schulz. Combining Degree of Interest Functions and Progressive Visualization. In *Short Paper Proceedings of the IEEE Visualization Conference (VIS'23)*, pages 251–255. IEEE, 2023. doi:10.1109/VIS54172.2023.00059. 118

[126] M. Hogräfer and H.-J. Schulz. Tailorable Sampling for Progressive Visual Analytics. *IEEE Trans. Vis. Comput. Graph.*, pages 1–13, 2023. doi:10.1109/TVCG.2023.3278084. 31, 97

[127] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Trans. Vis. Comput. Graph.*, 25(8):2674–2693, 2018. doi:10.1109/TVCG.2018.2843369. 146, 147

[128] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, E. Eisemann, B. Lelieveldt, and A. Vilanova. Cytosplore: Interactive Immune Cell Phenotyping for Large Single-Cell Datasets. *Computer Graphics Forum*, 35(3):171–180, 2016. doi:10.1111/cgf.12893. 11

[129] A. Hughes and L. Palen. Twitter adoption and use in mass convergence and emergency events. *International Journal of Emergency Management*, 6(3):248–260, 2009. doi:10.1504/IJEM.2009.031564. 124

[130] J. Hullman, M. Kay, Y.-S. Kim, and S. Shrestha. Imagining replications: Graphical prediction & discrete visualizations improve recall & estimation of effect uncertainty. *IEEE Trans. Vis. Comput. Graph.*, 24(1):446–456, 2017. doi:10.1109/TVCG.2017.2743898. 106

[131] J. R. Hullman, X. Qiao, M. Correll, A. Kale, and M. Kay. In Pursuit of Error: A Survey of Uncertainty Visualization Evaluation. *IEEE Trans. Vis. Comput. Graph.*, 25:903–913, 2019. doi:10.1109/TVCG.2018.2864889. 96, 105, 161, 162

[132] S. Huron, R. Vuillemot, and J.-D. Fekete. Visual Sedimentation. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2446–2455, 2013. doi:10.1109/TVCG.2013.227. 118, 121

[133] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of Data Exploration Techniques. In *International Conference on Management of Data*, SIGMOD '15, pages 277–281, New York, NY, USA, 2015. ACM. doi:10.1145/2723372.2731084. 9

[134] M. Imran, C. Castillo, F. Diaz, and S. Vieweg. Processing Social Media Messages in Mass Emergency: A Survey. *ACM Computing Surveys*, 47(4):67:1–67:38, 2015. doi:10.1145/2771588. 124

[135] M. Imran, C. Castillo, J. Lucas, P. Meier, and S. Vieweg. AIDR: artificial intelligence for disaster response. In *International Conference on World Wide Web*, WWW'14, pages 159–162, 2014. doi:10.1145/2567948.2577034. 124, 125

[136] M. Imran, P. Mitra, and C. Castillo. Twitter as a Lifeline: Human-annotated Twitter Corpora for NLP of Crisis-related Messages. In *International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1638–1643, 2016. 125

[137] T. Isenberg, P. Isenberg, J. Chen, M. Sedlmair, and T. Möller. A Systematic Review on the Practice of Evaluating Visualization. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2818–2827, 2013. doi:10.1109/TVCG.2013.126. 170

[138] ISO 5721-1. Accuracy (trueness and precision) of measurement methods and results - Part 1: General principles and definitions, 1994. URL `https://www.iso.org/obp/ui/#iso:std:11833:en`. 20

[139] A. C. Janes, D. A. Pizzagalli, S. Richardt, B. de B Frederick, A. J. Holmes, J. Sousa, M. Fava, A. E. Evins, and M. J. Kaufman. Neural substrates of attentional bias for smoking-related cues: An FMRI study. *Neuropsychopharmacology*, 35(12):2339–2345, 2010. doi:10.1038/npp.2010.103. 122

[140] A. Jena, U. Engelke, T. Dwyer, V. Raiamanickam, and C. Paris. Uncertainty visualisation: an interactive visual survey. In *2020 IEEE Pacific Visualization Symposium (PacificVis)*, IEEE Pacific Visualization Symposium, pages 201–205. IEEE, 2020. doi:10.1109/PacificVis48177.2020.1014. 94, 96, 105

[141] J. Jo, S. L'Yi, B. Lee, and J. Seo. ProReveal: Progressive Visual Analytics with Safeguards. *IEEE Trans. Vis. Comput. Graph.*, 27(7):3109–3122, 2021. doi:10.1109/TVCG.2019.2962404. 87, 88, 97, 105, 107, 113, 122, 160, 164

[142] J. Jo, J. Seo, and J.-D. Fekete. PANENE: A Progressive Algorithm for Indexing and Querying Approximate k-Nearest Neighbors. *IEEE Trans. Vis. Comput. Graph.*, 26(2):1347–1360, 2020. doi:10.1109/TVCG.2018.2869149. 31, 60, 61, 62, 63, 72, 77, 79, 98, 104, 140, 156, 157, 166, 167, 175

[143] J. Jo, F. Vernier, P. Dragicevic, and J.-D. Fekete. A Declarative Rendering Model for Multiclass Density Maps. *IEEE Trans. Vis. Comput. Graph.*, 25(1):470–480, 2019. doi:10.1109/TVCG.2018.2865141. 72

[144] S. Jo and I. Trummer. BitGourmet: Deterministic Approximation via Optimized Bit Selections. In *CIDR 2020, Online Proceedings*, pages 1–7, 2020. 38

[145] T. Joachims. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer, 2002. doi:10.1007/978-1-4615-0907-3. 141

[146] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*, 2017. doi:10.48550/arXiv.1702.08734. 140

[147] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 2016-03-11]. 50

[148] M. I. Jordan. Artificial intelligence—the revolution hasn't happened yet. *Harvard Data Science Review*, 1(1):1–9, 2019. doi:10.1162/99608f92.f06c6e61. 183

[149] D. Kahneman. *Thinking, fast and slow*. Macmillan, 2011. 165

[150] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of CHI*, CHI '11, pages

3363–3372, New York, NY, USA, 2011. ACM. doi:10.1145/1978942.1979444. 5

[151] Y.-A. Kang and J. Stasko. Examining the use of a visual analytics system for sense-making tasks: Case studies with domain experts. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2869–2878, 2012. doi:10.1109/TVCG.2012.224. 165

[152] Z. S. Karnin, K. J. Lang, and E. Liberty. Almost Optimal Streaming Quantiles Algorithms. *arXiv preprint arXiv:1603.05346*, abs/1603.05346, 2016. doi:10.48550/arXiv.1603.05346. 52, 53

[153] R. M. Karp. On-Line Algorithms Versus Off-Line Algorithms: How Much is it Worth to Know the Future? In *IFIP congress (1)*, volume 12, pages 416–429, 1992. 24

[154] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Mélancon. Visual Analytics: Definition, Process and Challenges. In *Information Visualization - Human-Centered Issues and Perspectives*, number 4950 in LNCS, pages 154–175. Springer, Aug. 2008. URL `http://hal-lirmm.ccsd.cnrs.fr/lirmm-00272779`. State-of-the-Art Survey. 13, 14, 73, 210

[155] A. Kerren and J. T. Stasko. Algorithm Animation. In *Software Visualization*, pages 1–15. Springer, 2002. doi:10.1007/3-540-45875-1_1. 18, 119, 121

[156] M. Khan, L. Xu, A. Nandi, and J. M. Hellerstein. Data tweening: incremental visualization of data transforms. *Proc. VLDB Endow.*, 10(6):661–672, 2017. doi:10.14778/3055330.3055333. 121

[157] H. Kim, J. Choo, C. Lee, H. Lee, C. Reddy, and H. Park. PIVE: per-iteration visualization environment for real-time interactions with dimension reduction and clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI'17, page 1001–1009. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.10628. 10

[158] G. Kindlmann and C. Scheidegger. An algebraic process for visualization design. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2181–2190, 2014. doi:10.1109/TVCG.2014.2346325. 163

[159] F. A. A. Kingdom and N. Prins. *Psychophysics: A Practical Introduction*. Elsevier Science, 2016. 163

[160] H.-K. Ko, J. Jo, and J. Seo. Progressive Uniform Manifold Approximation and Projection. In *EuroVis 2020 - Short Papers*. The Eurographics Association, 2020. doi:10.2312/evs.20201061. 72, 98

[161] M. Korporaal, I. T. Ruginski, and S. I. Fabrikant. Effects of Uncertainty Visualization on Map-Based Decision Making Under Time Pressure. *Frontiers in Computer Science*, 2, 2020. doi:10.3389/fcomp.2020.00032. 150, 160

[162] N. Kourtellis, G. De Francisci Morales, and F. Bonchi. Scalable online betweenness centrality in evolving graphs. In *International Conference on Data Engineering (ICDE)*, pages 1580–1581, May 2016. doi:10.1109/ICDE.2016.7498421. 57

[163] C. Kozyrkov. Your dataset is a giant inkblot test, 2019. URL `https://towardsdatascience.com/your-dataset-is-a-giant-inkblot-test-b9bf4c53eec5`. retrieved 04-

MAR-2020. 122

[164] G. E. Krasner and S. T. Pope. A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, Aug. 1988. 66

[165] J. F. Kruiger, A. Hassoumi, H.-J. Schulz, A. C. Telea, and C. Hurter. Multidimensional Data Exploration by Explicitly Controlled Animation. *Informatics*, 4(3):26:1–26:21, 2017. doi:10.3390/informatics4030026. 117

[166] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, Mar. 1951. doi:10.1214/aoms/1177729694. 95

[167] B. C. Kwon, M.-J. Choi, J. T. Kim, E. Choi, Y. B. Kim, S. Kwon, J. Sun, and J. Choo. Retainvis: Visual analytics with interpretable and interactive recurrent neural networks on electronic medical records. *IEEE Trans. Vis. Comput. Graph.*, 25(1):299–309, 2018. doi:10.1109/TVCG.2018.2865027. 147

[168] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale. Empirical studies in information visualization: Seven scenarios. *IEEE Trans. Vis. Comput. Graph.*, 18(9):1520–1536, 2011. doi:10.1109/TVCG.2011.279. 170

[169] J. Lazar, J. H. Feng, and H. Hochheiser. *Research methods in human-computer interaction*. Morgan Kaufmann, 2017. 163

[170] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. doi:10.1038/nature14539. 136

[171] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi:10.1162/neco.1989.1.4.541. 143

[172] F. Li, B. Wu, K. Yi, and Z. Zhao. XDB: approXimate DB. `https://initialdlab.github.io/XDB/`. Accessed on May, 19th, 2020. 37, 56

[173] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient Mini-Batch Training for Stochastic Optimization. In *International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 661–670, New York, NY, USA, 2014. ACM. doi:10.1145/2623330.2623612. 31, 104

[174] N. Li, V. van Unen, T. Höllt, A. Thompson, J. van Bergen, N. Pezzotti, E. Eisemann, A. Vilanova, S. M. Chuva de Sousa Lopes, B. P. Lelieveldt, and F. Koning. Mass cytometry reveals innate lymphoid cell differentiation pathways in the human fetal intestine. *Journal of Experimental Medicine*, 215(5):1383–1396, 03 2018. doi:10.1084/jem.20171934. 181

[175] E. Liberty. Simple and Deterministic Matrix Sketching. In *International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 581–588, New York, NY, USA, 2013. ACM. doi:10.1145/2487575.2487623. 61

[176] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007. doi:10.1073/pnas.0709640104. 60

[177] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2456–2465, Dec 2013. doi:10.1109/TVCG.2013.179. 9, 172

[178] D. Liu, W. Cui, K. Jin, Y. Guo, and H. Qu. Deeptracker: Visualizing the training process of convolutional neural networks. *ACM Trans. Intell. Syst. Technol.*, 10(1):1–25, 2018. doi:10.1145/3200489. 147

[179] Z. Liu and J. Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2122–2131, Dec 2014. doi:10.1109/TVCG.2014.2346452. 2, 5, 20, 33

[180] Z. Liu and J. Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2122–2131, 2014. doi:10.1109/TVCG.2014.2346452. 108, 110, 164

[181] S. B. Loeschke, M. Hogräfer, and H.-J. Schulz. Progressive Parameter Space Visualization for Task-Driven SAX Configuration. In *Proceedings of the International EuroVis Workshop on Visual Analytics (EuroVA)*, pages 43–47. Eurographics, 2020. doi:10.2312/eurova.20201085. 116

[182] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*, 2017. doi:10.48550/arXiv.1706.08840. 136

[183] V. Losing, B. Hammer, and H. Wersing. kNN classifier with self adjusting memory for heterogeneous concept drift. In *International Conference on Data Mining (ICDM)*, pages 291–300. IEEE, 2016. doi:10.1109/ICDM.2016.0040. 145

[184] V. Losing, B. Hammer, and H. Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018. doi:10.1016/j.neucom.2017.06.084. 145

[185] K.-L. Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Comput. Graph. Appl. Mag*, 29(6):14–19, Nov 2009. doi:10.1109/MCG.2009.120. 18

[186] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1: Statistics, pages 281–297. University of California Press, Berkeley, Calif., 1967. 85, 136

[187] A. Macy. Visualizing Sorting Algorithms. `https://bl.ocks.org/alexmacy/770f14e11594623320db1270361331dc`. Accessed: 2020-03-08. 115

[188] M. J. Mahoney. Publication prejudices: An experimental study of confirmatory bias in the peer review system. *Cognitive Therapy and Research*, 1(2):161–175, 1977. doi:10.1007/BF01173636. 120, 122

[189] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010. 49

[190] A. Mayorga and M. Gleicher. Splatterplots: Overcoming Overdraw in Scatter Plots. *IEEE Trans. Vis. Comput. Graph.*, 19(9):1526–1538, Sept 2013. doi:10.1109/TVCG.2013.65. 70

[191] A. McGregor. Graph Stream Algorithms: A Survey. *SIGMOD Rec.*, 43(1):9–20, May 2014. doi:10.1145/2627692.2627694. 61

[192] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426*, 2018. doi:10.48550/arXiv.1802.03426. 72, 136

[193] W. McKinney. pandas: a Foundational Python Library for Data Analysis and Statistics. *Python for High Performance and Scientific Computing*, pages 1–9, 2011. 50, 57

[194] P. McLachlan, T. Munzner, E. Koutsofios, and S. North. LiveRAC: Interactive Visual Exploration of System Management Time-Series Data. In *Proceedings of CHI*, CHI '08, pages 1483–1492, 2008. doi:10.1145/1357054.1357286. 168

[195] M. Meyer, T. Munzner, and H. Pfister. MizBee: A Multiscale Synteny Browser. *IEEE Trans. Vis. Comput. Graph.*, 15(6):897–904, 2009. doi:10.1109/TVCG.2009.167. 168

[196] M. Meyer, M. Sedlmair, P. S. Quinan, and T. Munzner. The nested blocks and guidelines model. *Information Visualization*, 14(3):234–249, 2015. doi:10.1177/1473871613510429. 170

[197] L. Micallef, H.-J. Schulz, M. Angelini, M. Aupetit, R. Chang, J. Kohlhammer, A. Perer, and G. Santucci. The Human User in Progressive Visual Analytics. In *Short Paper Proceedings of EuroVis'19*, pages 19–23. Eurographics Association, 2019. doi:10.2312/evs.20191164. 109, 122, 152

[198] P. M. Miller and N. S. Fagley. The Effects of Framing, Problem Variations, and Providing Rationale on Choice. *Personality and Social Psychology Bulletin*, 17(5):517–522, 1991. doi:10.1177/0146167291175006. 124

[199] R. B. Miller. Response Time in Man-computer Conversational Transactions. In *Proc. of the Fall Joint Computer Conference, Part I*, pages 267–277. ACM, 1968. doi:10.1145/1476589.1476628. 2

[200] T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019. doi:10.1016/j.artint.2018.07.007. 146

[201] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal Event Sequence Simplification. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2227–2236, 2013. doi:10.1109/TVCG.2013.200. 7

[202] J. Moody, D. McFarland, and S. Bender-deMoll. Dynamic Network Visualization. *American Journal of Sociology*, 110(4):1206–1241, 2005. doi:10.1086/421509. 75

[203] D. A. Moore and P. J. Healy. The trouble with overconfidence. *Psychological Review*, 115(2):502–517, 2008. doi:10.1037/0033-295X.115.2.502. 122

[204] D. A. Moore and D. Schatz. The three faces of overconfidence. *Social and Personality Psychology Compass*, 11(8):e12331, 2017. doi:10.1111/spc3.12331. 122

[205] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but Verify: Optimistic Visualizations of Approximate Queries for Exploring Big Data. In *Proceedings of CHI*, CHI '17, pages 2904–2915, New York, NY, USA, 2017. ACM. doi:10.1145/3025453.3025456. 77, 97, 105, 106

[206] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing Interactive Latency and Resolution Sensitivity for Scalable Linked Visualizations. In *Proceedings of CHI*, CHI '19, New York, NY, USA, 2019. ACM. doi:10.1145/3290605.3300924. 9, 36, 41, 78, 156, 172

[207] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the Black Box: Strategies for Increased User Involvement in Existing Algorithm Implementations. *IEEE Trans. Vis. Comput. Graph.*, 20(12):1643–1652, Dec 2014. doi:10.1109/TVCG.2014.2346578. 19, 22, 80, 98, 111, 117, 119, 135

[208] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications – Vol.1: VISAPP*, volume 2, pages 331–340, 2009. doi:10.5220/0001787803310340. 139

[209] M. Muja and D. G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36, 2014. doi:10.1109/TPAMI.2014.2321376. 61

[210] J. D. Mulder, J. J. van Wijk, and R. van Liere. A Survey of Computational Steering Environments. *Future Gener. Comput. Syst.*, 15(1):119–129, Feb. 1999. doi:10.1016/S0167-739X(98)00047-8. 28, 116

[211] T. Munzner. A Nested Process Model for Visualization Design and Validation. *IEEE Trans. Vis. Comput. Graph.*, 15(6):921–928, 2009. doi:10.1109/TVCG.2009.111. 170

[212] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, Aug. 2005. doi:10.1561/0400000002. 26, 157

[213] L. Nachmanson, R. Prutkin, B. Lee, N. H. Riche, A. E. Holroyd, and X. Chen. GraphMaps: Browsing Large Graphs as Interactive Maps. In *Proceedings of Graph Drawing*, pages 3–15. Springer, 2015. doi:10.1007/978-3-319-27261-0_1. 112

[214] New York Taxis & Limousine Commission. TLC Trip Record Data. `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`, 2024. 62, 71, 75, 122, 210

[215] D. T. Nguyen, K. Al-Mannai, S. R. Joty, H. Sajjad, M. Imran, and P. Mitra. Robust Classification of Crisis-Related Data on Social Networks Using Convolutional Neural Networks. In *International Conference on Web and Social Media, ICWSM*, pages 632–635, 2017. 124, 125

[216] S. C. North. Incremental layout in DynaDAG. In *Graph Drawing*, pages 409–418, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. doi:10.1007/BFB0021824. 73

[217] M. I. Norton, D. Mochon, and D. Ariely. The IKEA effect: When labor leads to love. *Journal of Consumer Psychology*, 22(3):453–460, 2012. doi:10.1016/j.jcps.2011.08.002. 131

[218] P. C. O'Brien and T. R. Fleming. A Multiple Testing Procedure for Clinical Trials. *Biometrics*, 35(3):549, Sept. 1979. doi:10.2307/2530245. 101

[219] T. O'Donoghue and M. Rabin. Present Bias: Lessons Learned and to Be Learned. *American Economic Review*, 105(5):273–279, 2015. doi:10.1257/aer.p20151085. 122

[220] F. Ofli, P. Meier, M. Imran, C. Castillo, D. Tuia, N. Rey, J. Briant, P. M. Rudd, F. Reinhard, M. Parkan, and S. Joost. Combining Human Computing and Machine Learning to Make Sense of Big (Aerial) Data for Disaster Response. *Big Data*, 4(1):47–59, 2016. doi:10.1089/big.2014.0064. 125

[221] L. Padilla, M. Kay, and J. Hullman. *Uncertainty Visualization*, pages 1–18. John Wiley & Sons, Ltd, 2021. doi:https://doi.org/10.1002/9781118445112.stat08296. 96, 105

[222] M. Panju. Iterative methods for computing eigenvalues and eigenvectors. *arXiv preprint arXiv:1105.1185*, 2011. doi:10.48550/arXiv.1104.1185. 60

[223] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. VerdictDB: Universalizing Approximate Query Processing. In *International Conference on Management of Data*, SIGMOD '18, pages 1461–1476, New York, NY, USA, 2018. ACM. doi:10.1145/3183713.3196905. 34, 155

[224] A. Patil, G. Richer, C. Jermaine, D. Moritz, and J.-D. Fekete. Studying Early Decision Making with Progressive Bar Charts. *IEEE Trans. Vis. Comput. Graph.*, 29(1):407–417, 2023. doi:10.1109/TVCG.2022.3209426. 52, 54, 77, 95, 97, 102, 105, 107, 159, 161

[225] P. P. Pebay. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical Report SAND2008-6212, Sandia National Laboratories, 9 2008. doi:10.2172/1028931. 60

[226] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195. 57, 60

[227] A. Perer and B. Shneiderman. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In *Proceedings of CHI*, CHI '08, pages 265–274, 2008. doi:10.1145/1357054.1357101. 168

[228] D. Petersohn, S. Macke, D. Xin, W. Ma, D. Lee, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, and A. Parameswaran. Towards Scalable Dataframe Systems. *Proc. VLDB Endow.*, 13(12):2033–2046, July 2020. doi:10.14778/3407790.3407807. 55, 56, 59

[229] E. Petraki, S. Idreos, and S. Manegold. Holistic Indexing in Main-memory Column-stores. In *International Conference on Management of Data*, SIGMOD '15, pages 1153–1166, New York, NY, USA, 2015. ACM. doi:10.1145/2723372.2723719. 56

[230] N. Pezzotti, J.-D. Fekete, T. Höllt, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Multiscale Visualization and Exploration of Large Bipartite Graphs. *Computer Graphics Forum*, 37(3):549–560, 2018. doi:10.1111/cgf.13441. 57, 162

[231] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE Trans. Vis. Comput. Graph.*, 24(1):98–108, 2017. doi:10.1109/TVCG.2017.2744358. 119, 147

[232] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical Stochastic Neighbor Embedding. *Computer Graphics Forum*, 35(3):21–30, 2016. doi:10.1111/cgf.12878. 72, 103, 136

[233] N. Pezzotti, B. P. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tSNE for progressive visual analytics. *IEEE Trans. Vis. Comput. Graph.*, 23(7):1739–1752, 2016. doi:10.1109/TVCG.2016.2570755. 31, 72, 98, 104, 136, 140, 146, 162, 166

[234] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *International Conference on Intelligence Analysis*, volume 5, pages 2–4, 2005. 164

[235] C. Plaisant, G. Grinstein, and J. Scholtz. Visual-Analytics Evaluation. *IEEE Trans. Vis. Comput. Graph.*, 29(3):16–17, 2009. doi:10.1109/MCG.2009.56. 149

[236] S. J. Pocock. When (Not) to Stop a Clinical Trial for Benefit. *JAMA*, 294(17):2228–2230, 11 2005. doi:10.1001/jama.294.17.2228. 101

[237] A. Pol, C. M. Jermaine, and S. Arumugam. Maintaining very large random samples using the geometric file. *The VLDB Journal*, 17(5):997–1018, 2008. doi:10.1007/s00778-007-0048-z. 62, 173

[238] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006. doi:10.1109/MCAS.2006.1688199. 113

[239] N. Potti and J. M. Pate. DAQ: A New Paradigm for Approximate Query Processing. *Proc. VLDB Endow.*, 8(9):898–909, 2015. 38

[240] M. Procopio, A. Mosca, C. E. Scheidegger, E. Wu, and R. Chang. Impact of Cognitive Biases on Progressive Visualization. *IEEE Trans. Vis. Comput. Graph.*, 28(9):3093–3112, 2022. doi:10.1109/TVCG.2021.3051013. 161, 165

[241] M. Procopio, C. Scheidegger, E. Wu, and R. Chang. Load-n-Go: Fast Approximate Join Visualizations That Improve Over Time. In *Proceedings of DSIA*, 2017. 39, 56, 118

[242] M. Procopio, C. Scheidegger, E. Wu, and R. Chang. Selective Wander Join: Fast progressive visualizations for data joins. *Informatics*, 6(1), 3 2019. doi:10.3390/informatics6010014. 31, 37, 39, 56, 155, 156

[243] Y. Qiu, Y. Wang, K. Yi, F. Li, B. Wu, and C. Zhan. Weighted Distinct Sampling: Cardinality Estimation for SPJ Queries. In *International Conference on Management of Data*, SIGMOD '21, pages 1465–1477. ACM, 2021. doi:10.1145/3448016.3452821. 155

[244] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org/`. 49

[245] M. Raasveldt and H. Mühleisen. DuckDB: An Embeddable Analytical Database. In *International Conference on Management of Data*, SIGMOD '19, pages 1981–1984, New York, NY, USA, 2019. ACM. doi:10.1145/3299869.3320212. 56

[246] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 144

[247] G. Ramalingam and T. Reps. A categorized bibliography on incremental computation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 502–510, 1993. doi:10.1145/158511.158710. 25

[248] V. Raveneau, J. Blanchard, and Y. Prié. Progressive sequential pattern mining: steerable visual exploration of patterns with PPMT. In *Visualization in Data Science (VDS at IEEE VIS 2018)*, Berlin, Germany, 2018. IEEE. 158

[249] G. Richer, A. Pister, M. Abdelaal, J. Fekete, M. Sedlmair, and D. Weiskopf. Scalability in Visualization. *arXiv preprint arXiv:2210.06562*, abs/2210.06562, 2022. doi:10.48550/arXiv.2210.06562. 70, 172

[250] M. Rocklin. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference*, pages 130 – 136, 2015. doi:10.25080/MAJORA-7B98E3ED-013. 55, 59

[251] C. Rohrdantz, D. Oelke, M. Krstajić, and F. Fischer. Real-Time Visualization of Streaming Text Data: Tasks and Challenges. In *Proceedings of TextVis*, 2011. 115

[252] R. Rosenbaum and B. Hamann. Progressive Presentation of Large Hierarchies Using Treemaps. In *Advances in Visual Computing*, pages 71–80. Springer, 2009. doi:10.1007/978-3-642-10520-3_7. 77, 111

[253] R. Rosenbaum and H. Schumann. Progressive refinement: more than a means to overcome limited bandwidth. In *Proceedings of VDA*, page 72430I. SPIE, 2009. doi:10.1117/12.810501. 115, 118, 121

[254] R. Rosenbaum, J. Zhi, and B. Hamann. Progressive parallel coordinates. In *2012 IEEE Pacific Visualization Symposium*, pages 25–32. IEEE, 2012. doi:10.1109/PacificVis.2012.6183570. 77, 115

[255] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental Learning for Robust Visual Tracking. *Int. J. Comput. Vision*, 77(1-3):125–141, May 2008. doi:10.1007/s11263-007-0075-7. 60, 83, 166

[256] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571. IEEE, 2011. doi:10.1109/ICCV.2011.6126544. 140

[257] K. Rudra, S. Banerjee, N. Ganguly, P. Goyal, M. Imran, and P. Mitra. Summarizing Situational Tweets in Crisis Scenario. In *ACM Conference on Hypertext and Social Media*, pages 137–147. ACM, 2016. doi:10.1145/2914586.2914600. 124

[258] S. Ruping. Incremental learning with support vector machines. In *International Conference on Data Mining (ICDM)*, pages 641–642, 2001. doi:10.1109/ICDM.2001.989589. 140, 141

[259] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive Neural Networks, 2016, 1606.04671. 145

[260] F. Rusu, C. Qin, and M. Torres. Scalable Analytics Model Calibration with Online

Aggregation. *IEEE Data Eng. Bull.*, 38(3):30–43, 2015. 41

[261] R. B. Rusu and S. Cousins. 3D is here: Point cloud library (PCL). In *International Conference on Robotics and Automation*, pages 1–4. IEEE, 2011. doi:10.1109/ICRA.2011.5980567. 140

[262] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003. doi:10.1137/1.9780898718003. 25

[263] D. Sacha, A. Stoffel, F. Stoffel, B. C. Kwon, G. Ellis, and D. A. Keim. Knowledge Generation Model for Visual Analytics. *IEEE Trans. Vis. Comput. Graph.*, 20(12):1604–1613, 2014. doi:10.1109/TVCG.2014.2346481. 114

[264] W. Saeed and C. Omlin. Explainable AI (XAI): A systematic meta-survey of current challenges and future opportunities. *Knowledge-Based Systems*, 263:110273, 2023. doi:https://doi.org/10.1016/j.knosys.2023.110273. 119

[265] P. Saraiya, C. North, and K. Duca. An Insight-Based Methodology for Evaluating Bioinformatics Visualizations. *IEEE Trans. Vis. Comput. Graph.*, 11(4):443–456, 2005. doi:10.1109/TVCG.2005.53. 168

[266] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Trans. Vis. Comput. Graph.*, 23(1):341–350, 2017. doi:10.1109/TVCG.2016.2599030. 89

[267] T.-W. Schmidt, F. Pellacini, D. Nowrouzezahrai, W. Jarosz, and C. Dachsbacher. State of the Art in Artistic Editing of Appearance, Lighting and Material. *Computer Graphics Forum*, 35(1):216–233, 2016. doi:10.1111/cgf.12721. 159

[268] B. Schneider, D. Jäckle, F. Stoffel, A. Diehl, J. Fuchs, and D. Keim. Integrating Data and Model Space in Ensemble Learning by Visual Analytics. *IEEE Transactions on Big Data*, 7(3):483–496, 2021. doi:10.1109/TBDATA.2018.2877350. 117

[269] T. Schreck, J. Bernard, T. von Landesberger, and J. Kohlhammer. Visual cluster analysis of trajectory data with interactive Kohonen maps. *Information Visualization*, 8(1):14–29, 2009. doi:10.1109/VAST.2008.4677350. 117, 120, 121

[270] H.-J. Schulz, M. Angelini, G. Santucci, and H. Schumann. An enhanced visualization process model for incremental visualization. *IEEE Trans. Vis. Comput. Graph.*, 22(7):1830–1842, July 2016. doi:10.1109/TVCG.2015.2462356. 53, 82

[271] H. Schwerdtfeger. *Introduction to linear algebra and the theory of matrices*. Noordhoff, 1950. 141

[272] D. Sculley. Web-scale K-means Clustering. In *International Conference on World Wide Web*, WWW '10, pages 1177–1178. ACM, 2010. doi:10.1145/1772690.1772862. 31, 60, 83

[273] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley, 4th edition, 2011. 139

[274] M. Sedlmair and M. Aupetit. Data-Driven Evaluation of Visual Quality Measures. *Computer Graphics Forum*, 34(3):201–210, June 2015. doi:10.1111/CGF.12632. 151

[275] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual Parameter Space Analysis: A Conceptual Framework. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2161–

2170, 2014. doi:10.1109/TVCG.2014.2346321. 116

[276] M. Sedlmair, P. Isenberg, D. Baur, M. Mauerer, C. Pigorsch, and A. Butz. Cardiogram: visual analytics for automotive engineers. In *Proceedings of CHI*, CHI '11, pages 1727–1736, 2011. doi:10.1145/1978942.1979194. 168

[277] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2431–2440, 2012. doi:10.1109/TVCG.2012.213. 167, 168

[278] S. C. Seow. *Designing and Engineering Time: The Psychology of Time Perception in Software*. Addison-Wesley Professional, 2008. 149, 164

[279] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017. doi:10.48550/arXiv.1705.08690. 136

[280] K. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, 1994. doi:10.1109/5.259423. 20

[281] B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *ACM Comput. Surv.*, 16(3):265–285, Sept. 1984. doi:10.1145/2514.2517. 2, 36, 110

[282] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343. IEEE, 1996. doi:10.1109/VL.1996.545307. 115, 177

[283] B. Shneiderman and C. Plaisant. Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies. In *Proceedings of the AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, pages 1–7, 2006. doi:10.1145/1168149.1168158. 167, 168

[284] S. Si, S. Kumar, and Y. Li. Nonlinear Online Learning with Adaptive Nyström Approximation. *arXiv preprint arXiv:1802.07887*, 2018. doi:10.48550/arXiv.1802.07887. 142

[285] F. Siddiqui, T. Höllt, and A. Vilanova. A Progressive Approach for Uncertainty Visualization in Diffusion Tensor Imaging. *Computer Graphics Forum*, 40(3):411–422, 2021. doi:10.1111/CGF.14317. 102

[286] D. J. Simons and R. A. Rensink. Change blindness: Past, present, and future. *Trends in cognitive sciences*, 9(1):16–20, 2005. doi:10.1016/j.tics.2004.11.006. 163

[287] M. M. Skeels, B. Lee, G. Smith, and G. G. Robertson. Revealing uncertainty for information visualization. *Inf. Vis.*, 9(1):70–81, 2010. doi:10.1057/ivs.2009.1. 96

[288] D. Smilkov, S. Carter, D. Sculley, F. B. Viégas, and M. Wattenberg. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788*, 2017. doi:10.48550/arXiv.1708.03788. 119, 121

[289] M. Sondag, B. Speckmann, and K. Verbeek. Stable Treemaps via Local Moves. *IEEE Trans. Vis. Comput. Graph.*, 24(1):729–738, 2018. doi:10.1109/TVCG.2017.2745140. 75

[290] D. Song and E. Golin. Fine-grain Visualization Algorithms in Dataflow Environments. In *Proc. of the 4th Conference on Visualization '93*, pages 126–133. IEEE, 1993. doi:10.1109/VISUAL.1993.398860. 73

[291] K. Starbird, L. Palen, A. L. Hughes, and S. Vieweg. Chatter on the Red: What Hazards Threat Reveals About the Social Life of Microblogged Information. In *ACM Conference on Computer Supported Cooperative Work*, pages 241–250. ACM, 2010. doi:10.1145/1718918.1718965. 124

[292] I. Steinwart. Support Vector Machines are Universally Consistent. *Journal of Complexity*, 18(3):768–791, 2002. doi:https://doi.org/10.1006/jcom.2002.0642. 140

[293] C. D. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Trans. Vis. Comput. Graph.*, 20(12):1653–1662, 2014. doi:10.1109/TVCG.2014.2346574. 11, 19, 21, 22, 73, 82, 105, 119, 121, 151, 157, 158, 168

[294] M. Stonebraker, P. Brown, J. Becla, and D. Zhang. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science and Engg.*, 15(3):54–62, May 2013. doi:10.1109/MCSE.2013.19. 55

[295] H. Strobelt, B. Hoover, A. Satyanaryan, and S. Gehrmann. LMdiff: A Visual Diff Tool to Compare Language Models. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 96–105, Nov. 2021. doi:10.18653/v1/2021.emnlp-demo.12. 147

[296] M. Sugiyama and M. Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT Press, 2012. doi:10.7551/mitpress/9780262017091.001.0001. 145

[297] Tableau Software. `https://www.tableau.com/`, 2003. Accessed: 2019-07-20. 36

[298] Tensorflow Playground. Github, 2016. URL `https://playground.tensorflow.org/`. 106

[299] A. K. Thomas and P. R. Millar. Reducing the Framing Effect in Older and Younger Adults by Encouraging Analytic Processing. *The Journals of Gerontology: Series B*, 67B(2):139–149, 2011. doi:10.1093/geronb/gbr076. 124

[300] TPC-H. `http://www.tpc.org/tpch/`, 2016. Accessed: 2021-11-02. 155

[301] C.-H. Tsai, C.-Y. Lin, and C.-J. Lin. Incremental and Decremental Training for Linear Classification. In *International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 343–352, New York, NY, USA, 2014. ACM. doi:10.1145/2623330.2623661. 142

[302] M. Tukan, C. Baykal, D. Feldman, and D. Rus. On coresets for support vector machines. In *International Conference on Theory and Applications of Models of Computation*, pages 287–299. Springer, 2020. doi:10.1007/978-3-030-59267-7_25. 142

[303] J. W. Tukey. *Exploratory data analysis*. Addison-Wesley series in behavioral science : quantitative methods. Addison-Wesley, Reading (Mass.), 1977. URL `http://opac.inria.fr/record=b1080310`. 4

[304] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. Designing Progressive and Interactive Analytics Processes for High-Dimensional Data Analysis. *IEEE Trans. Vis. Comput. Graph.*, 23(1):131–140, 2017. doi:10.1109/TVCG.2016.2598470. 60, 78, 80, 82, 83, 84, 116, 146, 156, 160, 168

[305] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J.-D. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive data science: Potential and challenges. *arXiv preprint arXiv:1812.08032*, 2018. doi:10.48550/arXiv.1812.08032. 132

[306] A. Tversky and D. Kahneman. The framing of decisions and the psychology of choice. *Science*, 211(4481):453–458, 1981. doi:10.1126/science.7455683. 123

[307] A. Ulmer, M. Angelini, J.-D. Fekete, J. Kohlhammer, and T. May. A Survey on Progressive Visualization. *IEEE Trans. Vis. Comput. Graph.*, 30(9):6447–6467, 2024. doi:10.1109/TVCG.2023.3346641. 81

[308] S. D. Urban, T. B. Abdellatif, S. W. Dietrich, and A. Sundermier. Delta abstractions: A technique for managing database states in runtime debugging of active database rules. *IEEE Trans. Knowl. Data Eng*, 15(3):597–612, 2003. doi:10.1109/TKDE.2003.1198393. 119, 121

[309] A. C. Valdez, M. Ziefle, and M. Sedlmair. Priming and anchoring effects in visualization. *IEEE Trans. Vis. Comput. Graph.*, 24(1):584–594, 2017. doi:10.1109/TVCG.2017.2744138. 122, 165

[310] A. C. Valdez, M. Ziefle, and M. Sedlmair. Studying biases in visualization research: Framework and methods. In *Cognitive Biases in Visualizations*, pages 13–27. Springer, 2018. doi:10.1007/978-3-319-95831-6_2. 165

[311] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL `http://jmlr.org/papers/v9/vandermaaten08a.html`. 72, 85, 95, 104

[312] S. van der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2):22–30, March 2011. doi:10.1109/MCSE.2011.37. 50

[313] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995. 49

[314] J. Vermeulen, K. Luyten, K. Coninx, and N. Marquardt. The Design of Slow-motion Feedback. In *Proceedings of DIS*, pages 267–270. ACM, 2014. doi:10.1145/2598510.2598604. 79, 112

[315] E. Vernier, M. Sondag, J. Comba, B. Speckmann, A. Telea, and K. Verbeek. Quantitative Comparison of Time-Dependent Treemaps. *Computer Graphics Forum*, 39(3):393–404, 2020. doi:https://doi.org/10.1111/cgf.13989. 75

[316] J. Vidal, P. Guillou, and J. Tierny. A Progressive Approach to Scalar Field Topology. *IEEE Trans. Vis. Comput. Graph.*, 27(06):2833–2850, June 2021. doi:10.1109/TVCG.2021.3060500. 151

[317] J. S. Vitter. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57,

Mar. 1985. doi:10.1145/3147.3165. 44, 60, 72

[318] D. Vohra. Apache Parquet. In *Practical Hadoop Ecosystem*, pages 325–335. Apress, 2016. doi:10.1007/978-1-4842-2199-0_8. 55, 62

[319] W. A. Wagenaar and G. B. Keren. Calibration of probability assessments by professional blackjack dealers, statistical experts, and lay people. *Org. Behavior and Human Dec. Proc.*, 36(3):406–416, 1985. doi:10.1016/0749-5978(85)90008-1. 120

[320] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained K-means Clustering with Background Knowledge. In *International Conference on Machine Learning (ICML)*, pages 577–584. Morgan Kaufmann, 2001. 117

[321] A. Wald. *Sequential Analysis*. John Wiley and Son, Inc, New York., 1947. 94, 101, 161

[322] E. Wall, L. M. Blaha, L. Franklin, and A. Endert. Warning, Bias May Occur: A Proposed Approach to Detecting Cognitive Bias in Interactive Visual Analytics. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 104–115. IEEE, 2017. doi:10.1109/VAST.2017.8585669. 122

[323] Z. Wan, W. Taha, and P. Hudak. Event-Driven FRP. In *Practical Aspects of Declarative Languages*, pages 155–172, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45587-6_11. 66

[324] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, and A. Gray. Human-AI Collaboration in Data Science: Exploring Data Scientists' Perceptions of Automated AI. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019. doi:10.1145/3359313. 132

[325] J. Waser, R. Fuchs, H. Ribicic, B. Schindler, G. Blöschl, and E. Gröller. World lines. *IEEE Trans. Vis. Comput. Graph.*, 16(6):1458–1467, 2010. doi:10.1109/TVCG.2010.223. 117, 120, 121

[326] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016. doi:10.1007/s10618-015-0448-4. 145

[327] B. P. Welford. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4(3):419–420, 1962. doi:10.1080/00401706.1962.10490022. 30, 51, 60

[328] A. Weller. Transparency: motivations and challenges. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 23–40. Springer, 2019. doi:10.1007/978-3-030-28954-6_2. 146

[329] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. URL `http://ggplot2.org`. 89

[330] H. Wickham. Bin-summarise-smooth: a framework for visualising large data. Technical report, had.co.nz, 2013. URL `https://vita.had.co.nz/papers/bigvis.pdf`. 52, 68, 70, 97

[331] H. Wickham, D. Cook, H. Hofmann, and A. Buja. Graphical inference for infovis. *IEEE*

Trans. Vis. Comput. Graph., 16(6):973–979, Nov 2010. doi:10.1109/TVCG.2010.161. 122

[332] L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag New York, Inc., 2005. 89

[333] L. Wilkinson, A. Anand, and R. Grossman. Graph-Theoretic Scagnostics. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, INFOVIS '05, page 21, USA, 2005. IEEE. doi:10.1109/INFOVIS.2005.14. 72

[334] M. Williams and T. Munzner. Steerable, Progressive Multidimensional Scaling. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 57–64. IEEE, IEEE, 2004. doi:10.1109/INFVIS.2004.60. 11, 73, 166

[335] W. Wright, D. Schroh, P. Proulx, A. Skaburskis, and B. Cort. The Sandbox for analysis: concepts and methods. In *Proceedings of CHI*, CHI '06, pages 801–810. ACM, 2006. doi:10.1145/1124772.1124890. 122

[336] S. Xu, X. Zhang, and S. Liao. A Linear Incremental Nyström Method for Online Kernel Learning. *International Conference on Pattern Recognition (ICPR)*, pages 2256–2261, 2018. doi:10.1109/ICPR.2018.8545404. 142

[337] T. B. Yacoubian, D. A. S. A. Thani, and M. Aupetit. The Role of a Facilitator in Co-Design Applications for Exploratory Analysis in Domains of High Complexity: The Case of MAHiCGO. *IEEE Access*, 9:38296–38317, 2021. doi:10.1109/ACCESS.2021.3063468. 110, 130

[338] M. A. Yalçin, N. Elmqvist, and B. B. Bederson. Cognitive Stages in Visual Data Exploration. In *Proceedings of BELIV*, pages 86–95. ACM, 2016. doi:10.1145/2993901.2993902. 117

[339] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In *Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 423–438. ACM, 2013. doi:10.1145/2517349.2522737. 52

[340] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing. In *International Conference on Management of Data*, SIGMOD '14, page 277–288, New York, NY, USA, 2014. ACM. doi:10.1145/2588555.2588579. 102

[341] W. Zeng, C. Lin, J. Lin, J. Jiang, J. Xia, C. Turkay, and W. Chen. Revisiting the Modifiable Areal Unit Problem in Deep Traffic Prediction with Visual Analytics. *IEEE Trans. Vis. Comput. Graph.*, 2020. doi:10.1109/TVCG.2020.3030410. 147

[342] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Trans. Vis. Comput. Graph.*, 23(8):1977–1987, Aug 2017. doi:10.1109/TVCG.2016.2607714. 2, 33, 97, 105, 159, 160, 164, 168

[343] J. Zhang. Modern Monte Carlo methods for efficient uncertainty quantification and propagation: A survey. *WIREs Computational Statistics*, 13(5):e1539, 2021. doi:https://doi.org/10.1002/wics.1539. 104

[344] K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach. In *International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 1425–1434, La Palma, Canary Islands, 21–23 Apr 2012. PMLR. URL `https://proceedings.mlr.press/v22/zhang12d.html`. 142

[345] F. Zhao, S. Maiyya, R. Wiener, D. Agrawal, and A. E. Abbadi. KLL$^{\pm}$ approximate quantile sketches over dynamic datasets. *Proc. VLDB Endow.*, 14(7):1215–1227, Mar. 2021. doi:10.14778/3450980.3450990. 53

[346] J.-J. Zhu and J. Bento. Generative adversarial active learning. *arXiv preprint arXiv:1702.07956*, 2017. doi:10.48550/arXiv.1702.07956. 146

[347] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996. doi:10.1609/aimag.v17i3.1232. 27

List of Figures

1.1 Icicle tree visualization of 10 million patients events aggregated . . 8
1.2 The Information Visualization Reference Model, adapted from [41] 13
1.3 The Visual Analytics Reference Model, adapted from [154] 14
1.4 Book chapters related to the data analysis pipeline. 14

3.1 OLA-RAW architecture 42
3.2 ProgressiveDB architecture 44

4.1 Progressive bar charts without and with uncertainty. 54
4.2 Responsive *t*-SNE steps of the MNIST dataset loaded in a skewed order. 63

5.1 Standard visual representations do not scale well. 71
5.2 A screenshot from the SampleAction system. 74
5.3 In the NYC Taxi dataset [214], the density map of taxis loaded progressively (left) becomes the top left point (right) after loading a data row representing a taxi that traveled to Florida with its meter on, creating a scale problem. 75
5.4 Trade-off between stability and quality in progressive visualizations. 76
5.5 The DimXplorer System. 83
5.6 Visual components in DimXplorer. 84
5.7 The user interface of InsightsFeed [16] 86
5.8 UI Elements in InsightsFeed. 87
5.9 The ProReveal System. 88

6.1 The three phases of PDA uncertainty. 93
6.2 The user interface of InsightsFeed [16] 95

7.1 User roles, Tasks, Focus, and Usage in PVA. 121
7.2 The AIDR dashboard prototype 127

8.1 Different users and expectations for progressive machine learning . 134

List of Tables

2.1 Design requirements for creating PVA systems 19
2.2 Summary of the different methods of computation relatable to PDA. 31
2.3 Summary of the determining characteristics of each method of computation related to PDA. 32

7.1 Comparison of the current AIDR system and its progressive improvement. 129

List of Definitions

1 Mathematical and Computer function 23
2 Online function 24
3 Iterative function 25
4 Incremental Function 26
5 Streaming function 27
6 Real-time function 27
7 Anytime function 28
8 Progressive function 29
9 Approximate Query Processing Function 30

10 PDA uncertainty 93
11 The PDA quality 93

Listings

3.1 Example: SQL Progressive Query 37
3.2 SQL ALTER QUERY 39
3.3 SQL PROGRESSIVE Query 46
3.4 SQL PROGRESSIVE VIEW and FUTURE 47

4.1 Example: Progressive Mean 51
4.2 Example: Progressive Median 52
4.3 Example: Steerable Progressive Median with Filtering 53
4.4 Example: Filling a Tensor 55

Alphabetical Index

Accuracy, 23, 27, 58
Accuracy and Precision, 156
Aggregation, 70
    decomposable, 46
Algorithm animation, 18
Algorithmic
    stability, 20
`ALTER QUERY`, 39
Alternatives, 113
Anchor, 112
Animated transition, 77
Anytime, 14, 17, 23, **27**
Apophenia, 122
Approximation, 26
AQP, 23, **29**
Arrays, 49
Asynchronous, 42, 43
Attention management, 90
AutoML, 137

Bias, 152
    Cognitive, 69, 78, 94, 107, 109, 124, 130, 131, 163, 165, 169
        Attention, 122
        Belief, 120
        Confirmation, 120, 122
        Framing, 123
        IKEA, 131
        Illusion, 165
        Illusory truth, 122
        Information, 123
        Overconfidence, 122
        Present, 122
        Self-generation, 131
    cognitive
        belief, 120
        confirmation, 120
        exaggerated expectation, 120
    Data, 143
    Distribution, 62
    Statistical, 102
    System, 12
Bootstrap, 38
Bound
    CPU, 44
    I/O, 44
Bounded time, 27
Broadcasting operations, 58

Chunks, 24, 27, 30, 31, 35, 42–46, 53, 173
CLT-bounds, 38
Competitive analysis, 156–158
Confidence interval, 52, 77, 93, 100, 101
Control
    Slow down, 112
Converge, 20
Convergence, 20

CPU-bound, 44
CSV, 43

Data distribution, 43
Data Frames, 49
Data Sketch, *see* Synopse
Data streaming, 52
Database
    state, 38
Decision-making, 51, 88, 165
Delta, 27, 37, 66, 68
Design requirement
    R01, 18–21, 32, 80, 151
    R02, 19, 22, 32, 80, 151, 157, 162, 167
    R03, 18–22, 32, 87, 111, 151, 162
    R04, 19, 22, 32, 58, 80, 112, 151
    R05, 19, 22, 112, 151, 159
    R06, 18, 19, 21, 22
    R07, 18–22, 88, 112, 153, 159, 160
    R08, 19, 20, 22, 32, 111
    R09, 18–21, 32, 100, 111, 153
    R10, 18–21, 32, 81, 100, 111, 159, 160
    R11, 18–21, 32, 81, 100, 159, 160
    R12, 19, 20, 32, 78, 79, 82, 158
    R13, 19, 20, 22, 78, 81, 82, 157, 162, 167
    R14, 19, 20, 22, 78, 82, 85
    R15, 19, 20, 22, 78, 82, 85, 87
    R16, 19, 20, 22, 32, 78, 79, 82, 87
Dimensionality reduction, 72
DimXplorer, 83
Disadvantages, 9
Disclosure, 77
Drill-down, 39, 45, 46

Eager, 15, 16
Election, 2
Exploratory Data Analysis, 1

Feature
    Engineering, 137
    Selection, 137
Filtering, 39, 70
FITS, 43
`FORK QUERY`, 40
Function
    anytime, 27
    approximate, 29
    incremental, 25
    iterative, 24
    mathematical, 23
    online, 24
    progressive, 28
    streaming, 26

Grammar of Graphics, 89
Graph algorithms, 57
Graph drawing, 57
Graphs, 49
Green computing, 4
Group-by, 39

History navigation, 77

I/O-bound, 44
Iid, **30**, 101, 103
In-Situ, 18, 155
In-Situ Databases, 34
Incremental, 23, **25**

InsightsFeed, 85
Interaction, 89
Interactivity, 110
Iterative, 23, **24**

Join, 39, 155
  Progressive, 155
  Wander, 155, 156
JSON, 43
Julia, 49

k-means, 25, 60, 67, 79, 83, 85, 97, 98, 104, 105, 136, 167, 176
k-Nearest Neighbors, 60, 61, 98, 99, 104, 136, 138, 140, 166

Latency, 16, 45, 46, 110
Lineage, 36
Linear algebra, 49
Long-lasting computation, 89
Loss, 106

Machine learning
  interactivity, 136
  steerability, 137
MATLAB, 49
Mental map, 77, 79, 112, 166, 175
Middleware, 37, 38, 45
Mini-batch, 31
Monitoring, 51, 109, 111

Networks, 49, 56

OLA, *see* Online aggregation
Online, 23, **24**
Online aggregation, 10, 17, 34, 181
Optimal stopping, 1, 94, 101, 152, 158, 179

Outlier, 107
Overplotting, 70
Overview, 70

Parallel, 40, 41, 50, 113, 117, 160
Partition, 45
PDA Quality, 92
Phase
  Chaotic, 93
  Converged, 93
  Converging, 93
Pipeline, 5, 14, 28, 49
Pivoting, 39
Posterior, 93
Precision and Recall, 21, 23, 156
Progress, 69
Progressive, 17, **28**
  computation, 28
  concurrent, 90
  middleware, 37
  quantum, 28, 30, 31, 84
  table, 45
  view, 45
Progressive-in-chunks, 21, 30, 53, 59–61, 68, 70, 96, 99, 107, 156, 166, 167
Progressive-in-iterations, 30, 31, 53, 57, 60, 68, 96, 97, 99, 107, 156, 166, 167
ProReveal, 87, 107, 160
PVA-Guards, 87
Python, 49

Quality, 28, 54, 58, 61, 86, 92–96, 98–101, 103–105, 107
  loss, 106
Quantum, 27, 28, 33, 37, 84

Queries
  Progressive, 46
Query
  alter, 39
  derived, 36
  fork, 40
  Join, 35
  latency, 45
  lineage, 36
  steering, 39, 45, 46, 155

R, 49
Random samples
  with replacement, 31
  without replacement, 30
Real-time, 23, **27**, 27, 68
Recall, 156
Reproducibility, 124, 157, 166
Roles, 109
  explorer, 111, 113
  observer, 111
  searcher, 111, 112
Roll-up, 39

Sampling, 30, 35, 70, 97
  bi-level, 43, 44
  bias, 107
  block-level, 43
  mini-batch, 31
  reservoir, 44
  with replacement, 31
  without replacement, 30, 38
Satisficing, **165**
Scalability, 70, 80
  number of categories, 70
  number of dimensions, 70
  number of items, 70
Search/gather, 17, 21, 157
Sensemaking, 164
Sequential analysis, 101
Server log, 5
Short-term memory, 106
Shuffling, **30**, 35, 43, 62, 151, 152, 172
Significance testing, 100
Speculative
  analysis, 90, 152, 160
  exploration, 41, 87, 90, 164
  query, 41
Stability, 12, 20, 58, 64, 80
State, 38
Static data, 17
Stationary, 21, **30**
Stationary process, 62, 145, 151, 152, 156, 157, 167
Steering, 2, 28, 39, 45, 46, 52, 69, 109, 111, 113, 151, 152, 179
  hyperparameter, 79
  query, 79
Strategy, 50, 64
Streaming, 14, 17, 23, **26**
Summarization, 72
Summary statistics
  algebraic, 68
  distributive, 68, 97
  holistic, 68, 97
Synopse, 52, 60, 68
  quantile, 52

t-SNE, 47, 62, 63, 72, 85, 86, 90, 95, 96, 98, 99, 104, 166, 167
Table, 49
  Join, 155

Telemetry, 5
Tensors, 49
Time engineering, 108, 110
Time window, 26, 28, 52
Time-quality trade-off, 1, 48, 102, 130, 142, 149, 151, 157, 160, 173, 178

Uncertainty, 23, 28, 52, 69, 73, 77, 85, 87–89, 92–100, 103, 105–107, 156
    Stability, 100
User
    focus, 109
    roles, 109, 111
    tasks, 109

Vectors, 49
Visual
    quality, 75, 77
    stability, 73
Visualization
    Aggregation, 70
    overplotting, 70
    overview, 70
    scalability, 70
    summarization, 72

What-if, 113

Progressive Data Analysis

We live in an era where data is abundant and growing rapidly; databases storing big data sprawl past memory, reach computation limits, and become increasingly distributed. Engineers are designing new hardware and software systems, with new storage management and predictive computation, to sustain this growth. Yet, while good for data at scale, these infrastructures do not support exploratory data analysis (EDA) effectively. EDA allows analysts to make sense of data with little or no known model. It is essential in many application domains, from network security and fraud detection to epidemiology and preventive medicine. Data exploration is done through an iterative loop where analysts interact with data through computations that return results, usually shown with visualizations, which in turn are interacted with by the analyst again. EDA calls for highly responsive systems: at 500 ms, users change their querying behavior; past five or ten seconds, users abandon tasks or lose attention. To address this problem, a new computation paradigm has emerged in the last decade: *Progressive Data Analysis*.

This book is an introduction to this new paradigm. It explains the main scientific and technical benefits of performing complex data analysis progressively on large data. It also introduces the challenges that it raises to become fully usable. These important issues concern research fields that are traditionally separate in computer science: databases, scientific computing, machine learning, visualization, statistics, and human-computer interaction. The book ends with a research agenda to help the scientific community converge on key research questions.

ISBN: 978-3-03868-270-7

9 783038 682707